VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

METODY KÓDOVÁNÍ PROBLÉMU V EVOLUČNÍM NÁVRHU KOMBINAČNÍCH OBVODŮ

PROBLEM ENCODING METHODS IN EVOLUTIONARY DESIGN OF COMBINATIONAL CIRCUITS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ADAM SEDLÁČEK**
AUTHOR

VEDOUCÍ PRÁCE — **prof. Ing. LUKÁŠ SEKANINA, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Sedláček Adam**
Program: Informační technologie
Název: **Metody kódování problému v evolučním návrhu kombinačních obvodů**
Problem Encoding Methods in Evolutionary Design of Combinational Circuits
Kategorie: Umělá inteligence

Zadání:

1. Seznamte se s problematikou genetického programování (GP) a jeho využitím pro návrh kombinačních obvodů. Zaměřte se na porovnání různých způsobů zakódování obvodů používaných v GP.
2. Seznamte se s dostupnými knihovnami, které implementují GP pro návrh obvodů.
3. Rozšiřte zvolenou implementaci GP (popř. vytvořte vlastní implementaci), pomocí které bude možné statisticky porovnat alespoň dva způsoby zakódování obvodů.
4. Na sadě testovacích problémů experimentálně porovnejte zvolené způsoby kódování obvodů pro GP.
5. Zhodnoťte dosažené výsledky a diskutujte možnosti pokračování projektu.

Literatura:

- Dle pokynů vedoucího.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Sekanina Lukáš, prof. Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 30. října 2020

Abstrakt

Práce porovnává dva odlišné přístupy k zakódování kombinačních obvodů při automatizovaném návrhu obvodů, který využívá evolučních algoritmů. Porovnání proběhlo mezi kartézským genetickým programováním (CGP) a obvodem reprezentovaným v algebraické normální formě (ANF). Obě metody byly demonstrovány nad sadou vybraných obvodů. Byla porovnána rychlost konvergence nalezení prvního plně funkčního řešení. Jako druhé kritérium hodnocení byla plocha na čipu. Pro urychlení hodnocení kvality obvodů bylo využito paralelní simulace. Implementace proběhla v programovacím jazyce C++ s využitím Boost knihovny. Výhody a nevýhody obou metod zakódování jsou pak shrnuty v závěru této práce.

Abstract

The thesis compares two different approaches to combinational circuit encoding for automated circuit design which uses evolutionary algorithms. The comparison was made between cartesian genetic programming and circuit represented in the algebraic normal form. Both methods were evaluated on a chosen set of circuits. The first test case criterion was the convergence of each particular method. The second optimization criterion was the area used on a chip. For accelerating the evaluation of fitness a parallel simulation was used. Implementation is in programming language C++ with Boost library. The pros and cons of both methods are summarised at the end of this work.

Klíčová slova

CGP, evoluční návrh kombinačních obvodů, genetické programování, kartézské genetické programování, porovnání zakódování kombinačních obvodů, umělá inteligence, ANF, algebraická normální forma

Keywords

CGP, evolutionary design of combinational circuits, genetic programming, cartesian genetic programming, comparison of circuit encodings, artificial intelligence, ANF, algebraic normal form

Citace

SEDLÁČEK, Adam. *Metody kódování problému v evolučním návrhu kombinačních obvodů*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Lukáš Sekanina, Ph.D.

Metody kódování problému v evolučním návrhu kombinačních obvodů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Prof. Ing. Lukáše Sekaniny, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Adam Sedláček
10. května 2021

Poděkování

Tímto bych chtěl poděkovat svému vedoucímu Prof. Ing. Lukáši Sekaninovi, Ph.D za rady při psaní této bakalářské práce.

Obsah

Kapitola 1

Úvod

Evoluční algoritmy (EA) jsou optimalizační algoritmy, které stochasticky prohledávají prostor kandidátních řešení. Jsou inspirovány Darwinovou evoluční teorií, kde se uplatňuje přirozený výběr (přežití) nejsilnějších jedinců do další generace. EA využívá selekce, mutace a rekombinace. Prohledávaný prostor může být obrovský. Konvenční techniky pak selhávají nebo ani není možné je aplikovat. Nalezené řešení nemusí nutně být nejlepší možné, ale hledáme dostatečně kvalitní řešení. Více o evolučních algoritmech a jejich fungování je v kapitole 2.

Do této kategorie spadá i návrh kombinačních obvodů. Známé konvenční způsoby jsou popsány v kapitole 4, a způsoby vyjádření logické funkce jsou popsány v kapitole 3.

Protože evoluční algoritmy nejsou pevně ustálené jako jeden daný postup, stále probíhá výzkum, čehož je důkazem i tato bakalářská práce. Byl vybrán jeden ze známých způsobů zakódování kombinačních obvodů, které používá metoda zvaná kartézské genetické programování (CGP). Tato metoda je popsána v kapitole 5. Jako druhý způsob byla vybrána algebraická normální forma (ANF), která je popsána v kapitole 6.

Způsob zvolené implementace je v kapitole 7, kde jsou popsány hlavní části vlastní implementace.

Metrika a zkoumané vlastnosti obvodů jsou blíže uvedeny v kapitole 8. Pro testování byla vybrána sada různorodých obvodů. Tyto dvě techniky (CPG a ANF), byly postaveny proti sobě a vyhodnoceny, tak aby se dalo zjistit, které zakódování je vhodnější, pro jaké obvody a další případné zefektivnění metody.

V závěru 9 lze nalézt všeobecné zhodnocení této práce a možnosti dalšího pokračování v bádání.

Kapitola 2

Evoluční algoritmy

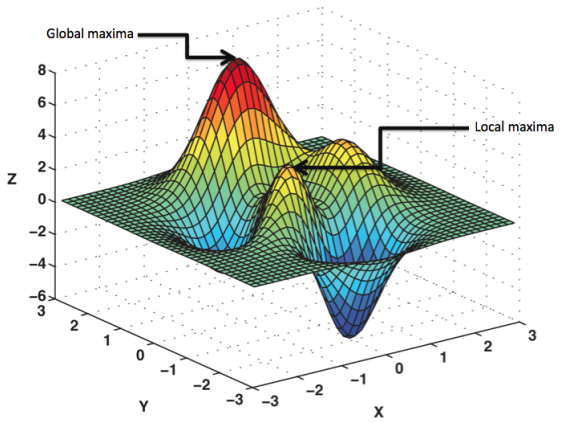
Evoluční algoritmy spadají pod kategorii umělé inteligence, kde přístup k řešení problému je inspirován v přírodě (Darwinovou evoluční teorií). Jedná se o rodinu algoritmů, které prohledávají stavový prostor stochasticky. Tento přístup se prokázal jako dostatečně účinný, a dokáže objevit nejen funkční, ale i řadu lepších řešení než konvenční techniky. Evoluční algoritmus může poskytnout inovativní řešení či řešení zcela nová, dosud neznámá [5].

Cílem je tedy nalézt řešení (jedince), které bude splňovat zadaná kritéria. Kritéria klidně můžeme kombinovat např. plochu na čipu (nižší příkon) a malé zpoždění. Je tedy nutné kandidátní řešení nějak vyhodnocovat, na to slouží účelová funkce (ohodnocení kvality).

V každém evolučním algoritmu jsou dva základní faktory, které jsou protichůdné. Jedním z nich je explorace, kterou je možné chápat jako průzkum, tj. objevení nových oblastí stavového prostoru. Druhým faktorem je explotace neboli využití doposud zjištěných slibných částí a následně jejich využití.

Aby evoluční algoritmus mohl správně pracovat, musí se vhodně navrhnout zakódování problému (genotyp). Dále návrhář musí řešit výběr do další generace, pomocí selekce, zároveň je zapotřebí podpořit i explorativní složku, čímž dosáhneme, aby algoritmus neuvázl v lokálním optimu. Z tohoto důvodu je potřeba zavést mutaci anebo rekombinací (křížení) respektive nějaký z genetických operátorů [5].

Evoluční algoritmy mají široké spektrum využití [13]. Od návrhu kombinačních obvodů, trysek, antén a dokonce i v umění.



Obrázek 2.1: Ukázka prostoru řešení, převzato z [8]

2.1 Reprezentace

Reprezentace problému je klíčová při řešení dané úlohy. Nevhodným zakódováním problému může dojít až k znemožnění funkčnosti evolučního algoritmu. V ideálním případě chceme dosáhnout efektivního zakódování, tak abychom jej nemuseli následně složitě převádět, případně kontrolovat podmínky validity, byť ne vždy je možné tomuto problému zabránit.

Chromozom je řetězec informací o daném jedinci (genotyp), taktéž nese informaci o jeho vlastnostech. Chromozom nám udává kód daného řešení v prostoru kandidátních řešení. Pokud máme dva stejné chromozomy, tak jsou rovněž stejné i vlastnosti [5]. Jedinci se následně řadí do populace. Chromozomy můžou být pevné délky, anebo proměnlivé délky.

Jedním z nejzákladnějších zakódování je binární zakódování (viz Tabulka 2.1), kde hodnoty nabývají pouze 0 a 1. V tomto případě je velikost pevně daná. Chromozom může být bez problému reprezentován jako vektor celých čísel, reálných čísel, nebo jako graf a podobně [4].

Chromozom A	0	1	1	0	0
Chromozom B	1	0	0	1	0

Tabulka 2.1: Ukázka zakódování binárních chromozomů

2.2 Selekce

Selekce je proces, při kterém se vybírají noví rodiče z populace (přežívají). Následně se pomocí genetických operátorů vytvoří nová generace, například pomocí mutace nebo křížení (případně jejich kombinací). Pokud jedinec přejde do nové populace beze změny. Jedná se o elitismus. Výběr je prováděn buďto zcela náhodně, anebo podle daného přístupu. Jedny z nejpoužívanějších metod jsou ruleta, turnajová selekce a selekce podle pořadí.

Selekční tlak (výběr jedinců do nové generace) je potřeba dobře zvolit, aby evoluční algoritmus, respektive populace nestagnovala ve vývoji [5].

2.3 Genetické operátory

Jedná se o operátory v genetickém algoritmu, které mění gen, popřípadě geny v chromozomu. Správně zvolené a nastavené operátory nám zajistí rychlou konvergenci algoritmu.

2.3.1 Mutace

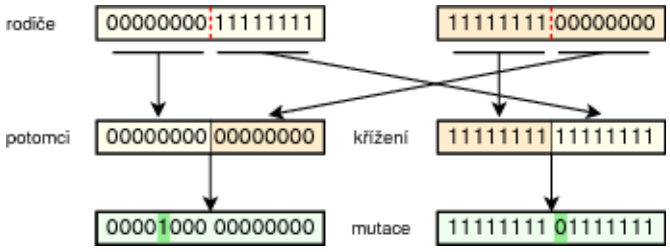
Operátor mutace mění náhodně gen nebo geny v chromozomu (ukázka možné mutace pro binární kódování na obr. 2.2). K velkým změnám by mělo docházet zpravidla méně často než k malým změnám. Mutace je závislá od daného zakódování. Například u binární zakódování se invertuje jeden bit (může i více) [4]. Existují široká variace mutací, můžeme například provést odstranění, nebo přidání hrany grafu, reálné hodnoty apod. Záleží na konkrétní variantě zakódování, a vhodnosti způsobu mutování.

Mutace je klíčová pro explorativní složku evolučního algoritmu, bez tohoto operátoru by došlo k uváznutí v lokálním optimu. Na druhou stranu příliš mnoho mutací taktéž škodí optimálnímu fungování algoritmu, kdy namísto nacházení lepšího jedince dochází k téměř náhodnému průchodu v prostoru řešení [5].

2.3.2 Křížení

Jeden ze základních operátorů je křížení. Pro křížení se musí vybrat alespoň dva rodiče (jedinci z populace), poté se náhodně zvolí bod pro křížení [4]. N genů se vezme z prvního rodiče, zbytek je doplněn z druhého rodiče. Jak může probíhat křížení je na obrázku 2.2.

Možnosti křížení jsou pestré. Od zvolení vícero rodičů, bodů až pro různé průměrování hodnot (u zakódování reálnými čísly), globální křížení apod. [5].



Obrázek 2.2: Operátor jednobodového křížení a mutace, převzato z [14]

2.4 Účelová funkce

Fitness funkce nám udává skóre pro každé řešení, jak moc je kvalitní. Opět platí, že musí být správně navržená, aby změny měly správný dopad na ohodnocení, pokud budou moc malé, nebude možné od sebe rozlišit, které řešení je lepší, a naopak, pokud budou změny moc velké, nemusíme nikdy dojít k funkčnímu řešení.

Hodnotu fitness lze získat jako porovnání vůči trénovací množině. Z výstup, který dostaneme od kandidátního řešení, je např. vypočtena průměrná odchylka mezi získanými a požadovanými hodnotami [5].

V evolučních algoritmech se využívá několik možností, jak hodnotu fitness získat:

- Hrubá fitness - udávána v hodnotách přirozených problémové doméně.
- Standardizovaná fitness – převádí Hrubou fitness do podoby, v které je stále menší numerická hodnota fitness žádanější (hodnota 0 znamená nejvyšší fitness).
- Přizpůsobená fitness - nejpoužívanější druh fitness. Vzniká jako obrácená hodnota součtu Standardizované fitness a čísla 1. Tento výpočet zabezpečí, že hodnoty fitness budou ležet v intervalu $<0, 1>$ (1 pro nejlepšího jedince, kde standardizovaná fitness $= 0$).
- Normalizovaná fitness - vytváří se jako podíl vhodnosti jedince a sumy vhodností všech jedinců populace. Tím se zabezpečí, že hodnota fitness leží v intervalu $<0, 1>$, hodnota fitness lepšího jedince je vyšší než hodnota horšího jedince a suma všech Normalizovaných fitness je rovna 1.

2.5 Multikriteriální optimalizace

Problémy v běžném životě většinou jdou v požadavcích proti sobě [3]. Například požadovaný obvod by měl v ideálním případě mít malou plochu, ale zároveň zpoždění mít také co nejmenší, což obvykle není možné současně splnit. Bohužel běžný optimalizační přístup je

většinou založený na fitness funkci, která zohledňuje pouze jedno kritérium. Zde nastává problém, že jedno kritérium ovlivňuje druhé kritérium protichůdně. Musí se tedy zvolit jistý kompromis, což je hlavním úkolem návrháře dané funkce, jaká kritéria upřednostnit. U některých problémů ani nelze určit, které řešení je kvalitnější. Pokročilé vícekriteriální optimalizační algoritmy proto nehledají pouze jedno řešení, ale tzv. Paretovu množinu řešení.

Při vícekriteriální optimalizaci se zohledňuje více aspektů v ohodnocení jedince. Existuje více přístupů, jak k danému vyhodnocení přistoupit. Jeden ze způsobů je přidělení váhy pro dané kritérium ve fitness funkci. Nevýhodou váhování je, že některá řešení budou nedosažitelná. Také není často zřejmé, jak správně váhy nastavit. Pokud chceme zohlednit více faktorů, tak se většinou využívají algoritmy, které zohledňují tzv. Pareto dominanci včetně hustoty zastoupení chromozomů v okolí. Jeden z algoritmů využívající právě tyto aspekty je NSGA [3].

2.6 Kanonický algoritmus

Přístup evolučního algoritmu je inspirovaný v přírodě [5]. Nová generace je vytvořena z předešlé populace vybraných jedinců. Velikost populace je většinou předem stanovená a pevná. Evoluční algoritmus musí mít rovněž i ukončovací kritérium. Nejčastěji vyčerpání počtu generací pro daný běh, případně nalezení daného řešení. Základní princip je popsán níže v bodech.

1. Vygenerování počáteční populace
2. Ohodnocení každého kandidátního jedince pomocí fitness funkce
3. Výběr vhodných jedinců ze stávající populace
4. Vytvoření nové populace z vybraných jedinců pomocí zvolených operátorů
5. Ohodnocení každého řešení nové populace
6. Pokud je splněna ukončující podmínka, kandidátní řešení s nejvyšší hodnotou fitness je výsledkem evolučního algoritmu. Pokud není splněna, pokračuje se bodem 3.

Kapitola 3

Formy vyjádření logické funkce

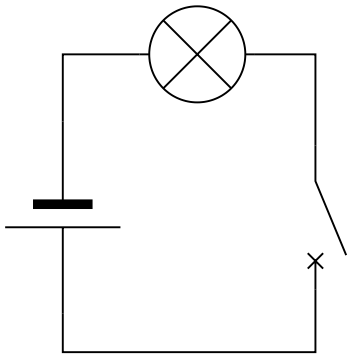
Matematické funkce dovolují popsat chování řady jevů, mj. i chování číslicových obvodů. Logická funkce přiřazuje výstupní logické hodnoty všem kombinacím vstupních logických proměnných (tj. log. 0 nebo log. 1). V této kapitole budou představeny různé možnosti popisu kombinačních obvodů. Logické obvody jsou složeny z logických hradel, která mají několik vstupů a jeden výstup. Proměnné nabývají hodnot 0 a 1. Chování hradla popisují elementární logické spojky, např. and, or, xor, not apod. U kombinačních obvodů výstup obvodu závisí pouze na aktuálních hodnotách vstupních proměnných.

3.1 Pravdivostní tabulka

Pravdivostní tabulka popisuje logickou funkci ve formě tabulky, kdy je každé kombinaci vstupních proměnných přiřazena pravdivostní hodnota, popř. více hodnot, pokud v jedné tabulce zapisujeme více logických funkcí. Potom první sloupec odpovídá prvnímu vstupu, druhý sloupec druhému vstupu atd. Obdobně je to pro sloupce s výstupy.

Příklad chování elektrického obvodu s žárovkou (svítí/nesvítí), spínačem (sepnut/nesepnut) a baterií (nabita/vybita), které lze popsat pravdivostní tabulkou, je na obr. 3.1 První vstup nám určuje, zda je sepnutý vypínač (může protékat elektrický proud). Druhý vstup určí, zda je nabita baterie. Pokud obě podmínky jsou splněny, tak žárovka bude svítit. V tomto konkrétním příkladě je použitá logická spojka *a*, kde oba vstupy musí mít log. hodnotu 1, aby výsledek (žárovka svítí) byl také log. 1. Pravdivostní tabulka pro všechny kombinace a schéma jsou na obr. 3.1. Nevýhodou pravdivostní tabulky je fakt, že počet řádků roste exponenciálně s počtem vstupů logické funkce, kterou popisuje. Proto se používá jen pro popis jednoduchých obvodů.

Nabitá baterie	Sepnutý vypínač	Žárovka svítí
0	0	0
0	1	0
1	0	0
1	1	1



Obrázek 3.1: Základní příklad pravdivostní tabulky s logickou spojkou a, včetně odpovídajícího schématu.

3.2 Dvouúrovňová reprezentace obvodů

Chování obvodu vyjádřené logickou funkcí, která sestává z k termů spojených operátorem s k vstupy nazýváme dvojúrovňovou reprezentaci logického obvodu. Pozn. do termu vstupuje vstupní signál (proměnná) buď v přímé nebo negované podobě [7].

3.2.1 Disjunktivní normální forma

Disjunktivní normální forma, označovaná také jako součet součinů, je jedním ze způsobů popisu v rámci dvojúrovňové reprezentace. Platí, že mezi termy jsou logické *NEBO*, a mezi literály je log. A, mimo jiné se dá využít ještě negace, ale pouze pro daný literál, nikoliv před terminálem [7].

Ukázka formule DNF:

$$f = (a \wedge b \wedge c) \vee (\neg a \wedge b \wedge c)$$

3.2.2 Konjunktivní normální forma

Pokud formule je v konjunktivní normální formě, platí že mezi termy je log. A, a mezi literály je log. *NEBO*. Také označována jako součin součtů [6].

Ukázka formule KNF:

$$f = (a \vee b \vee c) \wedge (\neg a \vee b \vee c)$$

Vyjádření DNF a KNF z pravdivostní tabulky

Z pravdivostní tabulky logické funkce f je možné sestavit DNK i KNF. V případě DNF je pro každou log. 1 na výstupu f sestaven součinový term, který obsahuje všechny vstupní proměnné, a tyto termy jsou spojeny logickým součtem. V případě KNF je pro každou log. 0 na výstupu f sestaven součtový term (s negovanými proměnnými) a tyto termy jsou spojeny logickým součinem [6].

Pokud máme dvě logické funkce f a g, které popisují stejnou logickou funkci, pak i jejich DNF budou identické. Toto platí i pro jejich KNF.

Příklad:

$$NDF : f = (a \wedge b \wedge \neg c) \vee (a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge c) \vee (a \wedge b \wedge c)$$

$$NKF : f = (a \vee b \vee c) \wedge (\neg a \vee b \vee c) \wedge (a \vee \neg b \vee c) \wedge (a \vee b \vee \neg c)$$

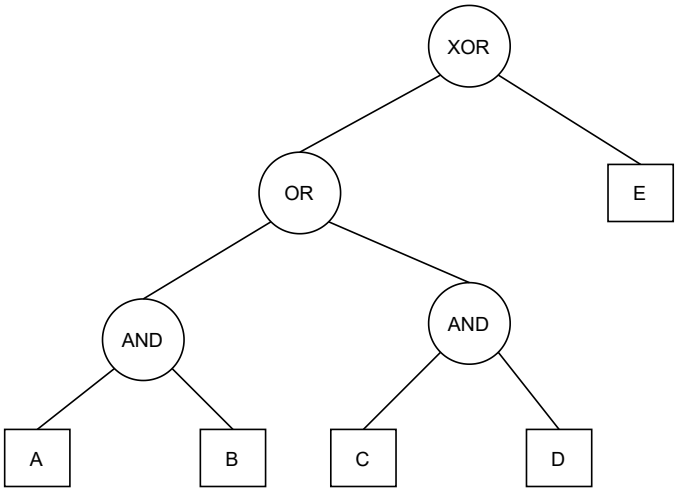
s	c	b	a	f	minterm	maxterm
0	0	0	0	0	-	$a + b + c$
1	0	0	1	0	-	$\bar{a} + b + c$
2	0	1	0	0	-	$a + \bar{b} + c$
3	0	1	1	1	$ab\bar{c}$	
4	1	0	0	0	-	$a + b + \bar{c}$
5	1	0	1	1	$a\bar{b}c$	
6	1	1	0	1	$\bar{a}bc$	
7	1	1	1	1	abc	

Obrázek 3.2: Pravdivostní tabulka pro ukázku DNF a KNF, převzato z [6]

3.3 Grafové reprezentace obvodu

Kombinační logický obvod může být modelován jako acyklický orientovaný graf, jehož vnitřní uzly jsou logická hradla a okrajové uzly jsou vstupu nebo výstupy. Použití této reprezentace potenciálně vede na nejkompaktnější realizaci na čipu, protože neuvažuje žádná další omezení na způsob propojení hradel [5].

Dále se pro reprezentaci logické funkce využívají binární rozhodovací diagramy, které jsou důležité zejména pro formální verifikaci.



Obrázek 3.3: Graf znázorňující $Y = ((A \wedge B) \vee (A \wedge B)) \oplus E$

Kapitola 4

Optimalizace kombinačních obvodů

Cílem minimalizace výrazů reprezentujících číslicové obvody je vytvořit co nejmenší (nejlevnější) obvod, který odpovídá zadané specifikaci. Nejčastěji se snažíme o redukci logických hradel (občas může dojít i redukci vstupů nebo výstupů) [7].

Optimalizace nemusí být jen z pohledu počtu hradel na čipu, ale odezvy, spotřeby a podobné. V těchto případech se pohybujeme ve víceúrovňové optimalizaci, tím pádem i v mnohem komplexnější přístupu k problému.

4.1 Konvenční způsoby

Pokud vyjádříme funkci (obvod) z pravdivostní tabulky (funkcí), tak se velmi často nejedná o minimální formu. Vzniklo mnoho konvenčních technik, jak rychle a efektivně minimalizovat počet termů pro danou funkci (zejména ve formě DNF nebo KNF).

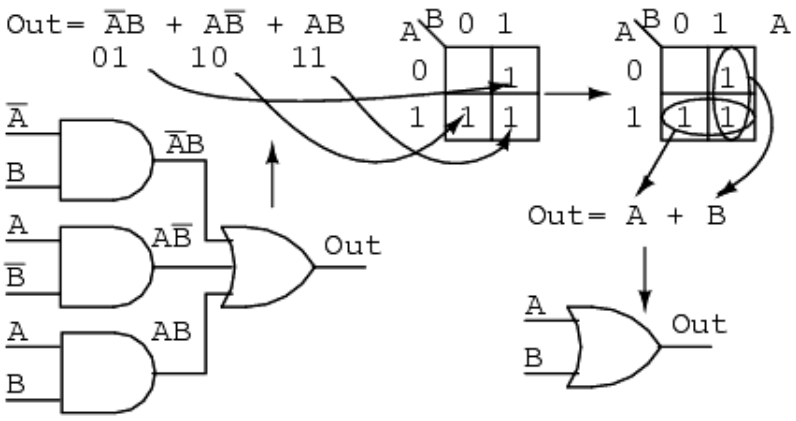
4.1.1 Karnaughova mapa

Karnaughova mapa je metoda, která funguje na principu přeuspořádání pravdivostní tabulky (nejčastěji) do dvojrozměrné matice (může být i vícerozměrná). Poté se v matici hledají smyčky, respektive booleovská sousednost, s cílem identifikovat redundantní proměnné či termy a odstranit je [7].

Pozici v Karnaughově mapě hodnoty určují dvě souřadnice, každá definovaná alespoň jednou vstupní proměnnou. Počet prvků mapy je 2^n, kde n je počet vstupních proměnných, obdobně počet řádků pravdivostní tabulky. Souřadnice pro hodnoty jsou v Grayově kódu, tudíž dvě sousední pole se vždy liší pouze o jednu proměnou. Sousedství buněk je definované cyklicky, tedy první buňka řádku (taktéž sloupce) je sousedem poslední [11] viz obr. 4.1.

Minimalizace se realizuje tak, že v Karnaughově mapě vyhledáme smyčky. Smyčka je tvořena jedničkami (případně X - *don't care*, pokud tím získáme větší oblast) a její velikost určují mocniny dvojky tedy 2, 4, 8, apod., zároveň by daná smyčka měla být co největší možná, s tím že může *přejít* přes okraj mapy (spojení prvního s posledním prvkem na řádku, respektive sloupci). Pokud zůstane jednička osamocena, přesto musí být posléze vyjádřena také, tj. nejedná se o podmnožinu jiného termu. Mezi jednotlivými smyčky je součet a mezi literály (log. 1) součin [7] [11].

Tato metoda je vhodná pro menší počet proměnných, a to ideálně do 5 včetně [2]. Zejména se hodí při ručním návrhu, kdy grafická podoba velmi napomáhá danému návrháři.



Obrázek 4.1: Karnaughova mapa s následnou minimalizací pomocí smyček, převzato z [1]

4.1.2 Quine McCluskey

Metoda Quine McCluskey je tabulková metoda spočívající v nalezení primárních implikantů a následným nalezení jejich minimální množiny [2]. Vznikla jako tabulková metoda pro ruční minimalizaci, tudíž tato metoda na dnešních počítačích umožňuje zpracovat maximálně nízké desítky proměnných (zhruba do 20 proměnných) kvůli své časové a prostorové složitosti (exponencionální). Vzniklo mnoho technik, které zvládají i větší počet proměnných [11].

4.1.3 Moderní způsoby optimalizace

Dnes se nejčastěji používají víceúrovňová reprezentace obvodů v tzv. Netlist[1] formátu. Algoritmy jsou součástí profesionálních nástrojů pro návrh obvodů a z akademických je nejpopulárnější ABC[2].

[1] https://en.wikipedia.org/wiki/Netlist
[2] https://people.eecs.berkeley.edu/~alanmi/abc/

Kapitola 5

Kartézské genetické programování

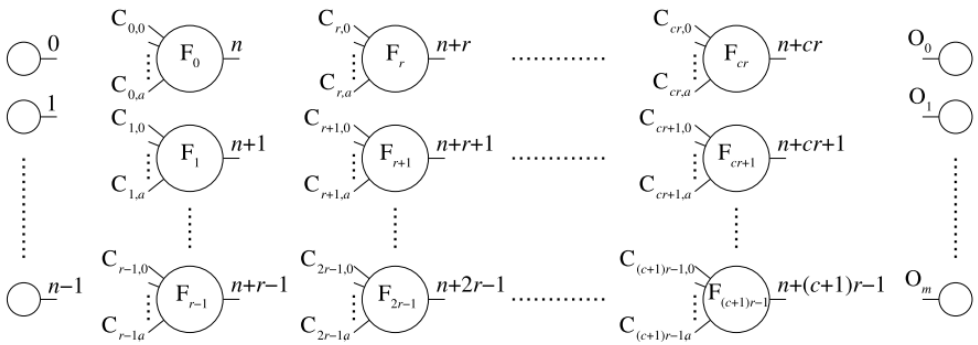
Tento přístup evolučního algoritmu se osvědčil hlavně při návrhu kombinačních obvodů [9], ale má využití i v jiných oblastech. Například ve strojovém učení, matematické regresi, evolučním umění apod. [13].

CGP využívá zakódování do acyklického grafu s pevným počtem uzlů (hradel) [14].

5.1 Reprezentace

CGP kóduje acyklicky orientovaný graf pomocí řetězce celočíselných hodnot. Jedinec je reprezentován jako mřížka, která má pevný počet řádků a sloupců. Velikost mřížky zůstává po celou dobu běhu konstantní, a vypočítá se jako $c \times r$, kde c je rovno počtu sloupců (column) a r je počet řádků (row). Každý uzel reprezentuje hradlo o n vstupech s právě jedním výstupem o. Uzel realizuje jednu z pevně daných funkcí nad danými vstupy [14, 9]. V případě této práce se jedná o dvouvstupová hradla, tedy do uzlu přichází maximálně 2 vstupy.

Nad takto definovanou maticí se posléze hledá vhodné propojení uzlů. Vnitřní propojení mezi uzly je řízeno tzv. L-back parametrem. Ten určuje, které uzly mohou být připojeny na vstup dalšího uzlu a to tak, aby nedocházelo ke smyčkám (zpětné vazbě). L-back se volí v rozmezí $l \in < 1, c >$. Pokud $l = 1$, tak se uzly napojují pouze na výstupy uzlů z předchozího sloupce [9]. Obecné zakódování acyklického grafu je na obr. 5.1.



Obrázek 5.1: Obecný graf pro CGP, převzato z [9]

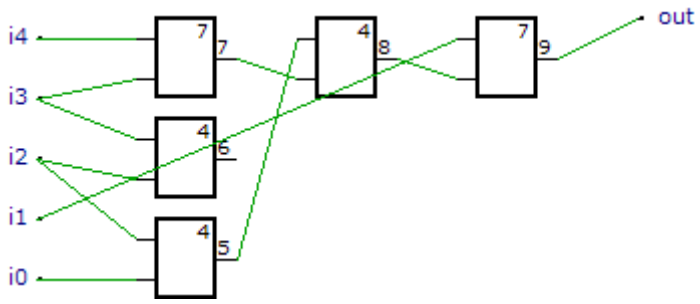
L-back parametr zásadně ovlivňuje zapojení hradel v obvodu (množinu vytvořitelných grafů a velikost stavového prostoru) [9]. V bodech níže je slovní popis, některých typických nastavení pro L-back.

- L-back = 1 – propojení je možné jen na jeden předchozí sloupec.
- L-back = max (počet sloupců) – není žádné omezení v propojení sloupců.
- L-back = max a počet řádků rovnající se 1 – jedná se o maximální možnou konektivitu.

5.1.1 Chromozom

Kandidátní řešení v CGP se zpravidla zakódovává do celočíselných řetězců délky $r \times c \times (n+1) + o$. Každý uzel má přiřazenu unikátní hodnotu, a to včetně primárních vstupů, které začínají na indexu s číslem 0. Zakódování jednoho uzlu se následně skládá z n celých čísel, které určují napojené vstupy (v této práci vždy $n = 2$), a z kódu logické funkce daného hradla, opět celočíselně například $0 = and, 1 = or, 2 = nand$ atd. Poslední část chromozomu tvoří k-tice velikostí shodnou s počtem primárních výstupů. Zakódování 5bit parity v CGP s parametry $c = 5, r = 1, n = 2, o = 1$:

$$(0,2,4)(2,3,4)(3,4,7)(5,7,4)(1,8,7)(9)$$



Obrázek 5.2: 5bit parita s nadbytečným hradlem č. 6, ($c = 5, r = 1$, přeuspořádáno dle zpoždění)

Vlevo označené i_i jsou vstupy. Uvnitř bloku je index pro realizovanou funkci, na výstupu unikátní index daného hradla (uzlu).

5.2 Mutace

Při experimentech s obvody v CGP se zjistilo, že křížení je nadbytečné, proto CGP využívá pro generování nové populace pouze operátor mutace [9]. Parametr mutace určuje, kolik alel se bude mutovat, nikoliv jaká je procentuální šance, že dojde k mutování daného jedince. Pokud tedy máme $mutace = 2\%, r = 1, c = 100, n = 2$ celkem se tedy bude mutovat 6krát (100 uzlů pronásobeno počtem prvků v uzlu a následně vypočteny 2 %) náhodně zvolená alela. Může se mutovat logická funkce daného hradla, například index *and* funkce se změní na index *or* funkce. Další možností mutace je jeden ze dvou vstupů daného uzlu (hradla), zde se musí přihlédnout k legálnosti daného propojení, a mutace zapojení výstupů.

Vliv mutace může být negativní, neutrální nebo pozitivní. Pokud nastane pozitivní mutace, tak se zvyšuje kvalita daného jedince. Negativní je přesný opak, kdy dochází ke zhoršení jedince. Aby nedocházelo k velké degradaci populace vlivem negativní mutace [13], tak se velmi často zavádí elitismus 2.2. Neutrální mutace hraje významnou roli při hledání funkčního řešení [9]. Neutrální mutace nemá dopad na fitness jedince, protože zasáhne buď nevyužitá hradla, nebo změní propojení použitých hradel, ale bez dopadu na logickou funkci obvodu. Posloupnost neutrálních mutací následovaná adaptivní (pozitivní) mutací může mít velký dopad na fenotyp i fitness jedince.

5.3 Fitness funkce

Při návrhu fitness funkce je potřeba zohlednit, jak náš daný problém vyhodnotíme. U obvodů se nabízí porovnávat s pravdivostní tabulkou. Hodnota fitness pak odpovídá počtu shod s předem zadanými výstupními hodnotami pro všechny vstupní kombinace [14]. Pokud máme např. $i = 4$ vstupy a $o = 2$ výstupy, pak počet vstupních kombinací je $2^i = 2^4 = 16$. Maximální fitness hodnota tedy může být $o \cdot 2^i = 2 \cdot 16 = 32$.

V této práci je využita fitness funkce, která upřednostňuje nižší hodnotu. Plně funkční obvod má tím pádem $fitness = 0$. Stejné přístup ohodnocení byl v této práci využit pro obě zakódování.

Referenční bity	0	1	1	0	0
Jedinec	0	1	0	0	1

Tabulka 5.1: Ukázka rozdílných bitů

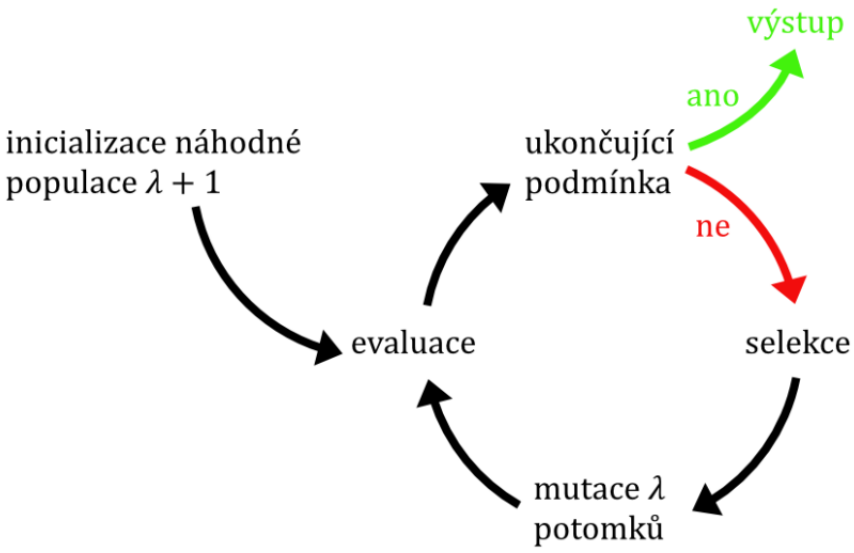
5.4 Evoluční strategie

Pro prohledávání stavového prostoru je využita evoluční strategie, která je typická zejména pro kartézské genetické programování [9], ale osvědčila se i pro obvody reprezentované algebraickou normální formou.

Algoritmus je založený na evoluční strategii $\lambda + 1$, λ udává počet mutovaných jedinců v generaci, většinou $\lambda = 4$. Kromě nově vzniklých jedinců je $+$ označení pro původního rodiče (elitismus). Z této populace se pak následně vybere nejlepší jedinec a ten přejde do další generace beze změny [5].

Mutace v tomto případě neudává procentuální šanci na zmutování, ale kolik genů chromozomu se bude měnit [9].

1. Vygenerování $\lambda + 1$ náhodných jedinců (chromozomů) pro inicializaci populace.
2. Ohodnocení všech jedinců populace pomocí fitness funkce.
3. Nalezení nejlépe ohodnoceného jedince — nej-nižší fitness (pokud jich existuje víc, vybere se ten, který nebyl rodičem předchozí generace).
4. Vygenerování λ potomků pomocí operátoru mutace aplikovaného na nejlepšího nalezeného jedince.
5. Nejlepší nalezený jedinec společně s jeho λ potomky tvoří novou populaci.
6. Ohodnocení všech jedinců populace pomocí fitness funkce.
7. Není-li splněna podmínka ukončení, pokračuje se krokem 3.



Obrázek 5.3: Algoritmus evoluce, respektive evoluční strategie $\lambda + 1$.

5.5 Použitá vícekriteriální optimalizace

V sekci 2.5 je nastíněno, že existuje více možností, jak optimalizovat s využitím dvou a více kritérií.

V této práci je využit dvojstupňový přístup. Princip tohoto přístupu spočívá v tom, že prvně najde plně funkční obvod a poté až následně optimalizuje plochu daného obvodu. Redukce hradel probíhá na základě mutování rodiče. Nová generace je tvořena pouze mutací původního plně funkčního obvodu. Pokud dojde k mutaci, která nezhoršuje funkčnost obvodu, a zároveň má menší plochu na čipu (než ostatní stejně kvalitní obvody v generaci), tak takový obvod je vybrán jako nový rodič.

Velikost	NOT	AND	OR	XOR	NAND	NOR	XNOR
Hradlo	1.40	2.34	2.34	4.69	1.87	2.34	4.69

Tabulka 5.2: Velikost použitých hradel je μm^2 a odpovídá 45 nm výrobnímu procesu. Převzato z [15].

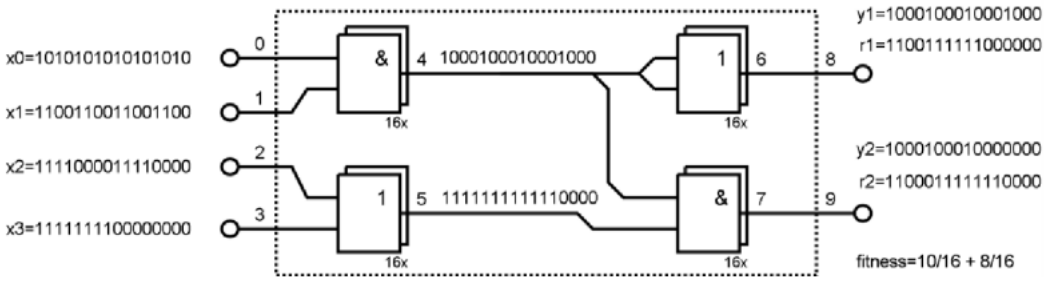
Plocha je vypočtena na základě zjištění počtu jednotlivých logických hradel, poté se vynásobí koeficientem, který udává, jakou plochu logický člen zabírá. Suma všech ploch logických hradel je výsledná plocha obvodu na čipu.

Pokud tedy máme obvod, který má 5 AND hradel, 4 OR hradla, a jedno NOT hradlo, tak výsledná plocha se vypočte jako:

$$5 \cdot 2.34 + 4 \cdot 2.34 + 1 \cdot 1.40 = 22.46 \mu m^2$$

5.6 Paralelní simulace

Pro výpočet fitness hodnoty obvodu je třeba získat jeho odezvu pro všechny vstupní kombinace. Pro představu obvod s $i = 10$ vstupy znamená vyhodnotit $2^i = 2^{10} = 1024$ vstupních vektorů. Proto při simulaci obvodů se velmi často využívá tzv. paralelní simulace, kdy jsou vstupní vektory vhodně zakódovány a výpočet probíhá pomocí vícebitových logických operací (obr. 5.4). Kdy vektor n bitů se vyhodnotí v jednom průchodu. Záleží na architektuře procesoru, nicméně dnes téměř každý procesor podporuje instrukční sadu AVX, kde velikost jednoho bitového vektoru je 256 nebo 512 bitů, tudíž dochází k masivnímu urychlení při vyhodnocení. Pokud uvážíme předchozí příklad, tak pro obvod s 10 vstupy je potřeba vyhodnotit 1024 jednotlivých bitů, ale s využitím vektoru bitů o velikosti 256, je potřeba provést simulaci obvodu pouze 4krát, což činní 256násobné zrychlení oproti simulaci po jednom vstupním vektoru.



Obrázek 5.4: Paralelní simulace pro obvod se 4 vstupy, převzato z [13]

Kapitola 6

Evoluce obvodů reprezentovaných v ANF

Algebraická normální forma (ANF) jeden ze způsobů reprezentace logické funkce. Oproti CGP, které dovoluje prakticky libovolné propojení hradel, zavádí značné restrikce.

6.1 Reprezentace

V případě ANF kódování pracujeme s obvody reprezentovanými v tzv. algebraické normální formě. Formule sestávají z termů, které jsou mezi sebou exkluzivně sečteny. Termy pak sestávají z literálů, které jsou spojeny konjunkcí. Literál může být přímý nebo negovaný [10]. Umožňuje i vybírat vstupní proměnné pro jednotlivé termy, přičemž maximální počet proměnných je fixním parametrem.

6.1.1 Chromozom

Počet literálů v daném termu je vždy roven počtu vstupů. Počet termů pak závisí na tom, jakou funkci chceme realizovat. Například pokud bychom se snažili popsat 5ti bitovou paritu pouze se třemi termy, pak takovou formuli nelze sestavit. Dalším zásadním parametrem je arita, která udává, kolik vstupů může být v jeden moment aktivních v jednom termu. Term s aritou například 3 je ekvivalentní se třivstupovým hradlem *AND*. Obvod s m výstupy modelujeme pomocí m formulí. Je zřejmé, že některé logické funkce budou v ANF zakódovány úsporně, jiné zase velmi nevýhodně.

Jako příklad uvažme obvod s $n = 3$ vstupy a $m = 2$ výstupy, jehož chování popisují formule:

$$y_0 = x_1x_2 \oplus x_1\bar{x_2}x_3 \oplus x_3$$
$$y_1 = x_1x_2 \oplus \bar{x_1}\bar{x_2}\bar{x_3}$$

Za předpokladu, že je ve formuli možné použít $t = 3$ termy, tento obvod bude zakódován řetězcem:

$$y_0 = 110\ 1\text{-}11\ 001$$
$$y_1 = 011\ \text{-}1\text{-}1\text{-}1\ 000$$

Kde 0, 1 a -1 popisují nepoužitou proměnnou, přítomnou proměnnou a negovanou proměnou. Délky chromozomu je potom $m \cdot n \cdot t = 2 \cdot 2 \cdot 3 = 12$ genů. Pomocí zavedení arity je možné redukovat počet použitých proměnných v termu.

Pro obvod (parita 5b), který je zakódován v ANF, kde parametry byly zvoleny $t = 5, arita = 1, n = 5, m = 1$ následující:

$$y_0 = x_0 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_4$$

Pokud by se jednalo o chromozom, tak reprezentace může vypadat následovně:

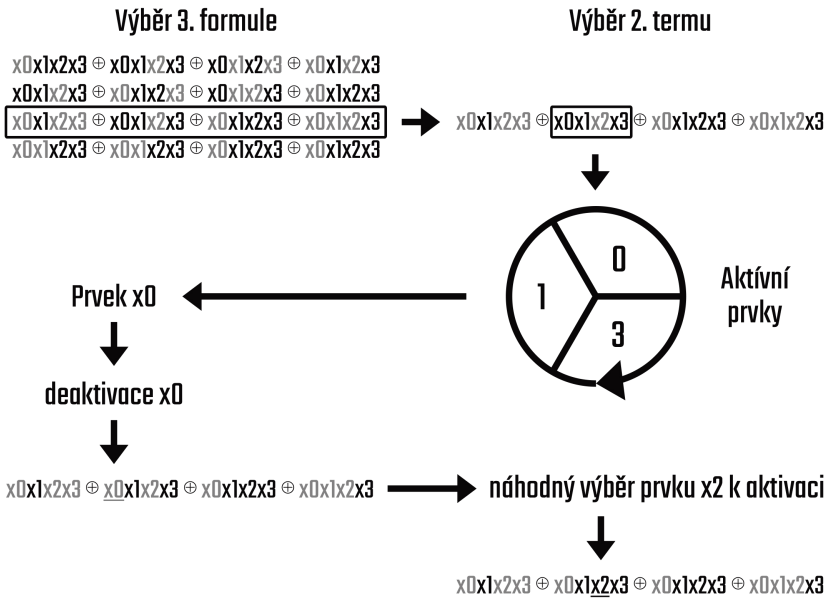
$$y_0 = 10000\ 01000\ 00100\ 00010\ 00001$$

6.2 Mutace

V případě zakódování pomocí ANF mutace není tak volná jako v CGP. Mutace probíhá náhodnou změnou stavu proměnné podílející se na výpočtu (negace, deaktivace, přítomnost). Výběr se uskuteční mezi formulemi, poté následuje výběr indexu vstupu a zjistí se validita změny. Pokud by se jednalo o porušení parametru arity, tak se v termu, který spadá do rozsahu indexu. Vybere náhodně aktivní prvek (negovaný nebo přítomnost) a provede se deaktivace. Díky tomu získáme možnost změny na vygenerovaném indexu.

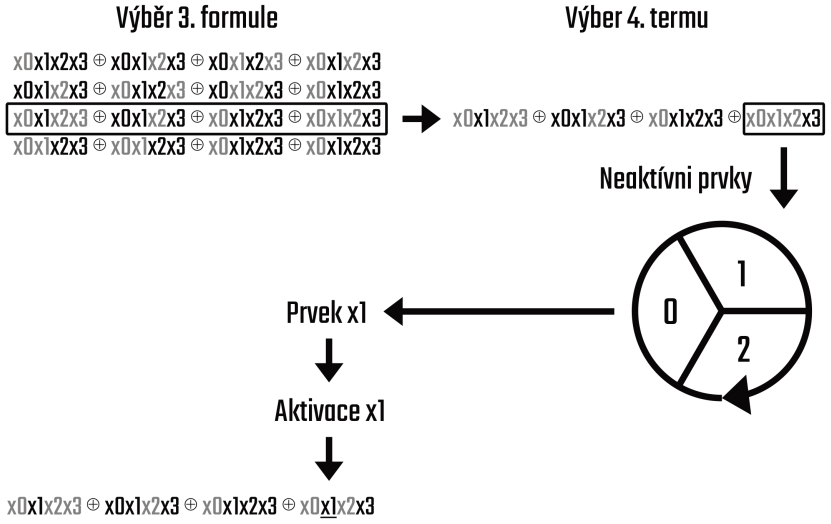
Parametr mutace určuje. Kolikrát proběhne mutace, tj. při $m = 5$, tak bude mutováno 5krát.

V příklade na obr. 6.1, kde parametry jsou $t = 4, arita = 3, n = 4, m = 4$ se bude mutovat jeden prvek. Náhodně byla vybrána třetí formule, následně byl vybrán druhý term, jenže, zde by se porušilo omezení arity (vznikly by 4 aktivní prvky), tím pádem bude náhodně vybráno z aktivních prvků x_0, x_1, x_3, který se deaktivuje a poté může být prvek x_2 aktivován.



Obrázek 6.1: Ukázka mutace v případě porušení parametru arity.

V příklade na obr. 6.2, kde jsou parametry stejné jako pro 6.1, tak se při dalším mutování vybrala opět třetí formule. Nyní, ale byl vybrán čtvrtý term, protože parametr arity nebyl porušen (pouze jeden aktivní prvek z maximálních 3). Proto byl vybrán prvek x_1, který se následně aktivuje.



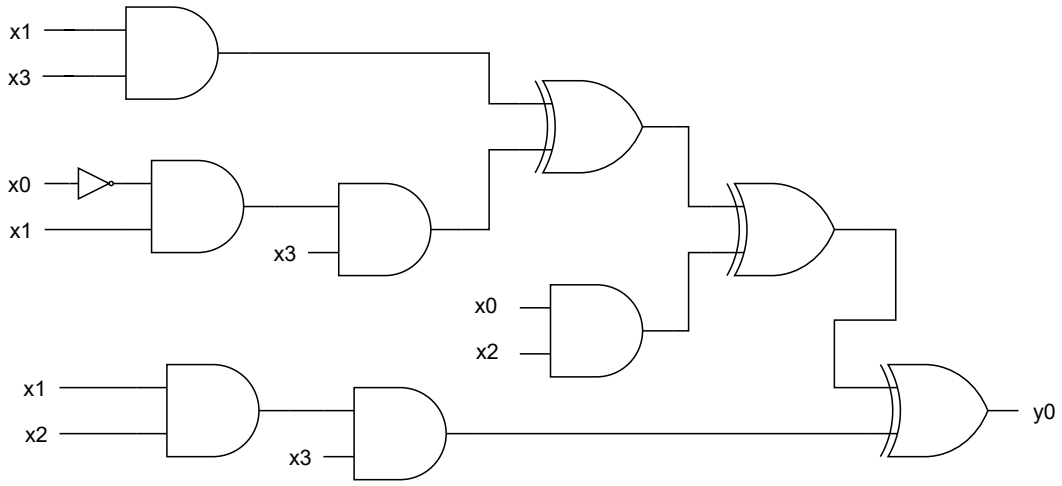
Obrázek 6.2: Ukázka mutace, pokud se nejedná o porušení arity.

6.3 Výpočet fitness

Fitness je obdobně vypočtena jako v případě CGP. Vůči referenčním vektoru bitů, kde nulový rozdíl znamená nalezení plně funkčního řešení (obvodu). Taktéž bylo využito paralelní simulace pro urychlení evaluace.

Při vyhodnocování funkčnosti obvodu dochází k jeho převedení na dvouvstupová hradla. Postupuje se zleva doprava. Mezi literály je provedena logická funkce AND. Poté co se vyhodnotí všechny aktivní prvky v termu, opět zleva doprava se provádí vyhodnocení termů, tentokrát s logickou funkcí XOR. Vztahy 6.1 popisují plně funkční 2b + 2b sčítačku. První formule je překreslena do odpovídajícího schématu pomocí dvouvstupových hradel na obr. 6.3. Parametry byly nastaveny jako počet termů = 4, arita = 3.

$$\begin{aligned} y_0 &= x_1x_3 \oplus \bar{x_0}x_1x_3 \oplus x_0x_2 \oplus x_1x_2x_3 \\ y_1 &= x_1x_2\bar{x_3} \oplus x_0x_1 \oplus x_0\bar{x}_1x_3 \oplus \bar{x_2}x_3 \\ y_2 &= x_0x_1\bar{x}_3 \oplus \bar{x_1} \oplus \bar{x_3} \oplus \bar{x_1}x_2\bar{x_3} \end{aligned} \tag{6.1}$$



Obrázek 6.3: Schéma zapojení pomocí dvouvstupových hradel odpovídající první formuli (řádku) ve vztahu 6.1.

6.4 Algoritmus s redukcí plochy

Při hledání kandidátních řešení se osvědčil stejný algoritmus jako u CGP, který je popsaný v sekci 5.4. Pro redukci plochy je taktéž využit stejný princip, který je popsaný v sekci 5.5 i s velikostí logických hradel na čipu. Nicméně obvod zakódovaný pomocí ANF metody je do jisté míry značně redundantní, díky tomu se nabízí různé optimalizace na výpočet plochy, které jsou blíže popsaný v implementační sekci 7.8.

Kapitola 7

Implementace

Tato kapitola se zabývá implementačními detaily jednotlivých částí. Prvně je zde popsán výběr programovacího jazyka, pomocí kterého byla implementace realizována, využité knihovny a nástroje. Na závěr je popis datových struktur a odpovídající diagram tříd pro každé ze zakódování.

7.1 Výběr programovacího jazyka

Před samotnou implementací je potřeba si rozmyslet výběr vhodného programovacího jazyka. Při výběru vhodného jazyka je nutné zvážit několik kritérií. Je vhodné vybírat jazyk, který má dobrou dokumentaci, ideálně má dostatečně velkou komunitu vývojářů, ale hlavně širokou podporu knihoven, které přijdou vhod. Z těchto důvodů většinou přijdou na výběr *populární* jazyky, jako jsou Java[1] nebo Python[2].

Oba výše zmíněné jazyky mají vše zmiňované. Dokonce se dají najít různé implementace evolučních algoritmů. Nicméně u CGP se většinou jedná o ne příliš dokumentovaný kód, případně nesplňuje požadovanou funkcionalitu, tj. rozšíření by bylo zbytečně časově náročné. Pokoušel jsem se najít i implementaci, která by odpovídala potřebám ANF, takovou se mi bohužel nepodařilo najít. Proto jsem se rozhodl obě řešení implementovat sám.

Java je objektově orientovaný jazyk, kód napsaný v Javě se dá lehce přenášet mezi systémy. Obdobně je to u Pythonu. Bohužel oba jazyky využívají abstraktní mezivrstvu. To přináší řadu výhod i nevýhod. Jednou z nevýhod je, že kusy kódu jsou přeloženy do strojového kódu až poté, co daná metoda nebo funkce je za běhu zavolána. Tím se do značné míry zpomaluje samotný běh programu. U Pythonu je k tomu dynamicky typovaný jazyk, tudíž se za běhu ještě musí ověřovat správnost typu (zpomalení). Oba jazyky díky své popularitě mají řadu modulů, které jsou napsány v C nebo C++, a daný jazyk slouží jako pouhé rozhraní[3] (např. Keras[4]). Vyvstává otázka, jestli by mělo smysl implementovat program v Pythonu (případně v Javě), i přes fakt, že řada modulů značně urychlí běh. Z tohoto důvodu jsem se raději rozhodl zvolit přímo C/C++. Programovací jazyk C++ je nadmnožinou C (s jistými rozdílnostmi), k tomu na rozdíl od C podporuje různé přístupy programovaní, podporuje OOP, generika apod. Proto jsem se rozhodl implementaci napsat v programovacím jazyce C++ ve standardu 17.

[1] https://go.java/

[2] https://www.python.org/

[3] Rozhraní (interface) označuje v informatice zařízení, program nebo formát, zajišťující správnou komunikaci a přenos dat mezi odlišnými zařízeními nebo programy.

[4] https://keras.io/

7.2 Použité knihovny a nástroje

C++ ve svém standardu podporuje datový typ *bitový vektor*[5] (bitset). Tento datový typ zastřešuje i logické operace nad vektory bitů. Pro paralelní simulaci v kap. 5.6 by poskytovaná funkcionalita byla dostatečná. Bohužel nevýhoda spočívá ve faktu, že počet bitů, které budou uloženy ve vektoru, je třeba znát předem při překladu, což bylo přípustné při prvotním návrhu. Bohužel s počtem přibývajících obvodů tento fakt začne být značně na obtíž. Z tohoto důvodu jsem se rozhodl pro využití knihovny Boost[6], která poskytuje dynamický bitový vektor[7].

Protože C++ je kompilovaný jazyk, musíme jej překládat pomocí kompileru. Projekt má více souborů, čímž vznikla potřeba nástroje, který nám kompilaci více souborů bez zbytečných potíží usnadní. Proto jsem se rozhodl využít nástroje CMake[8] pro jednoduchý překlad a správu projektu.

7.3 Struktura projektu

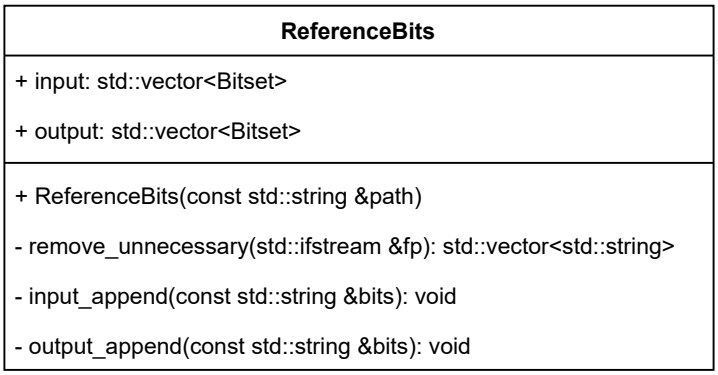
Struktura projektu je klíčová, pokud chceme kód do budoucna nějak udržovat, případně rozšiřovat. Proto jsem zvolil typické rozřazení souborů. Projekt je rozdělen do následujících souborů a složek:

- Složka src - Zdrojové soubory programů (*.hpp, *.cpp).
- Složka data - Soubory popisující obvody ve formátu PLA.
- Složka build - Složka, která vznikne po překladu projektu. Obsahuje spustitelné binární soubory.
- README.md - Obsahuje základní informace o programu (popis, instalace, spuštění).
- Makefile - Soubor určuje postup utility make při překladu a definuje závislosti mezi zdrojovými soubory.
- CMakeLists.txt - Soubor popisující pravidla sestavení, obsahu a cíle pro podadresáře.

7.4 Referenční bity

Pro reprezentaci vektoru bitů byl vytvořen vlastní datový typ *ReferenceBits*, jenž je využíván v obou reprezentacích. Pravdivostní tabulka obvodu je načtena ze souboru (trénovací množina vstupních vektorů), který je ve formátu PLA[9]. Datový typ drží vektory[10] bitů vstupů a výstupů (viz diagram tříd 7.1). Protože následně se v kódu velmi často využívá referenčních bitů (evaluace, výpočet fitness), tak je předáván pouze pomocí reference, aby nevznikala zbytečná režie kopírováním.

[5] https://en.cppreference.com/w/cpp/utility/bitset
[6] https://www.boost.org/
[7] https://www.boost.org/doc/libs/1_36_0/libs/dynamic_bitset/dynamic_bitset.html
[8] https://cmake.org/
[9] https://www.engr.colostate.edu/ECE571/class_materials/Mentor/Split/node32.html
[10] https://en.cppreference.com/w/cpp/container/vector



Obrázek 7.1: Diagram tříd datového typu ReferenceBits.

7.5 Třída CGP

Pro obvod reprezentovaný CGP metodou jsem vytvořil přímo třídu *Circuit CGP*. Výhoda třídy spočívá v jednoduché implementaci a budoucí rozšíření funkcionality by také nemělo být obtížné. Následně tato třída podporuje potřebné metody (výpočet fitness, evaluace, tisk ve formátu pro CGPviewer[11]). Každý obvod drží vektor buněk (hradel viz obr 7.2 *cell*). Každá instance buňky obsahuje informace o svých vstupech a vykonávané funkci, na základě, kterých vyprodukuje výstup ve formě bitsetu (při evaluaci). Více detailů implementace je vidět v diagramu 7.2.

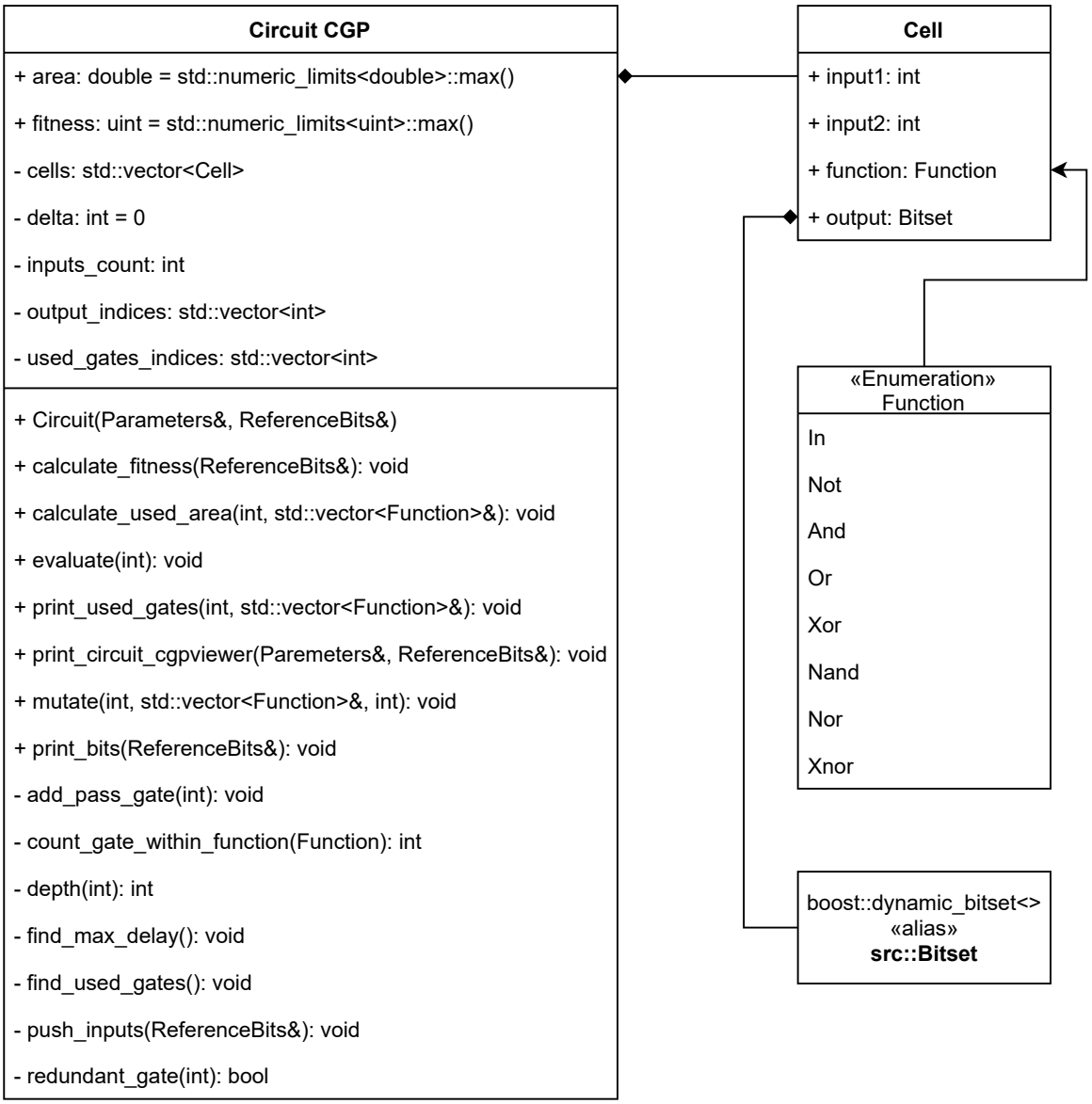
7.6 Výpočet plochy CGP

Výpočet plochy jsem implementoval jako rekurzivní průchod přes uzly, které se přímo podílejí na výpočtu (od výstupů ke vstupům). Každý navštívený uzel je uložen do vektoru použitých hradel (index). Pokud do uzlu je přiveden jeden a ten samý vstup dvakrát, tak v případě některých logických funkcí (AND, OR) se uzel nezapočte do plochy, protože se jedná pouze o *propojku*. Tabulka 7.1 znázorňuje vstup A a A' (jeden vstup).

A	A'	$\wedge$	$\vee$
0	0	0	0
1	1	1	1

Tabulka 7.1: Pravdivostní tabulka pro AND a OR hradlo se stejným vstupem A.

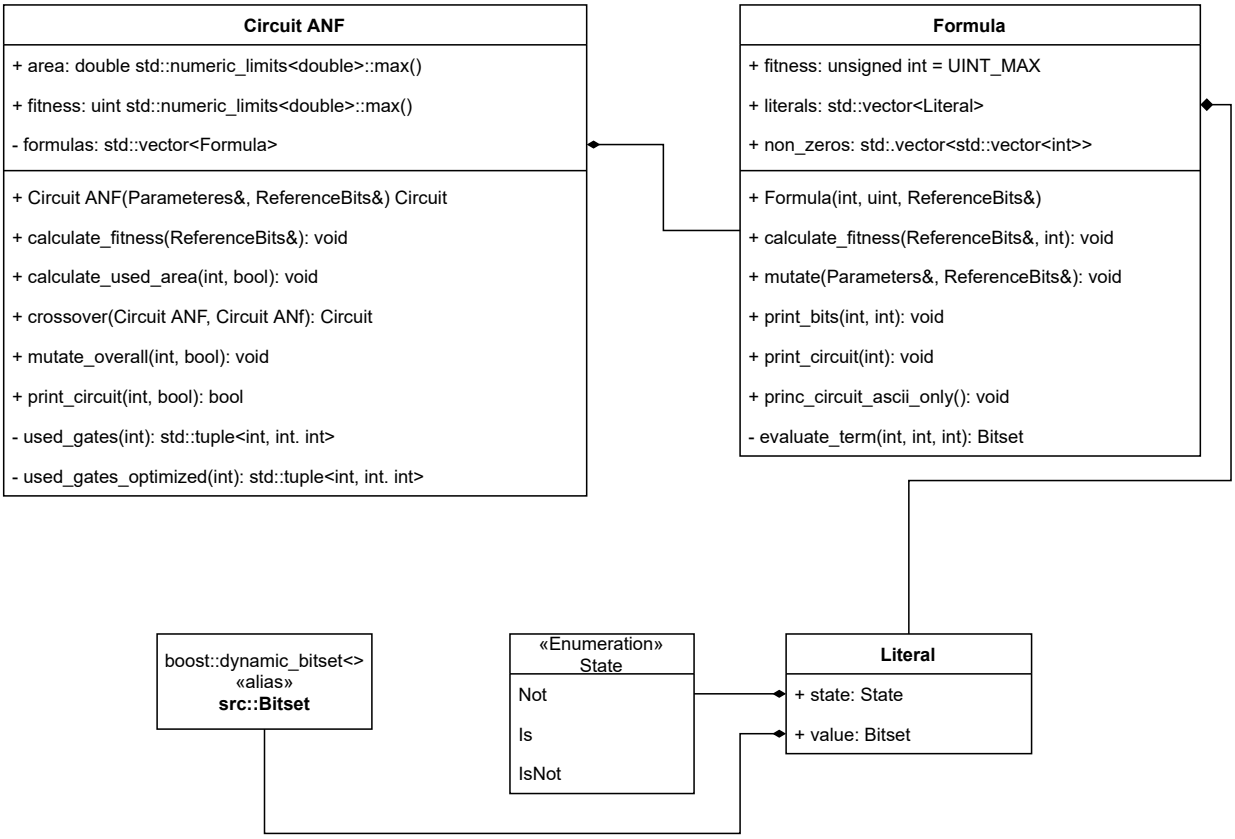
[11] Odkaz na sadu pomocných nástrojů http://www.fit.vutbr.cz/~vasicek/cgp/tools/.



Obrázek 7.2: Diagram tříd s komponenty potřebnými pro obvod zakódovaný v CGP.

7.7 Třída ANF

Zakódování obvodů metodou ANF je řešeno pomocí třídy *Circuit ANF*. Tato třída poskytuje propojení mezi formulemi, jako výpočet celkové fitness obvodu, výpočet plochy apod. Jedna instance v sobě drží několik instancí třídy *Formula* ve vektoru (jeden řádek rovnice odpovídá jedné formuli). Metody a vztahy tříd jsou vidět na diagramu 7.3.

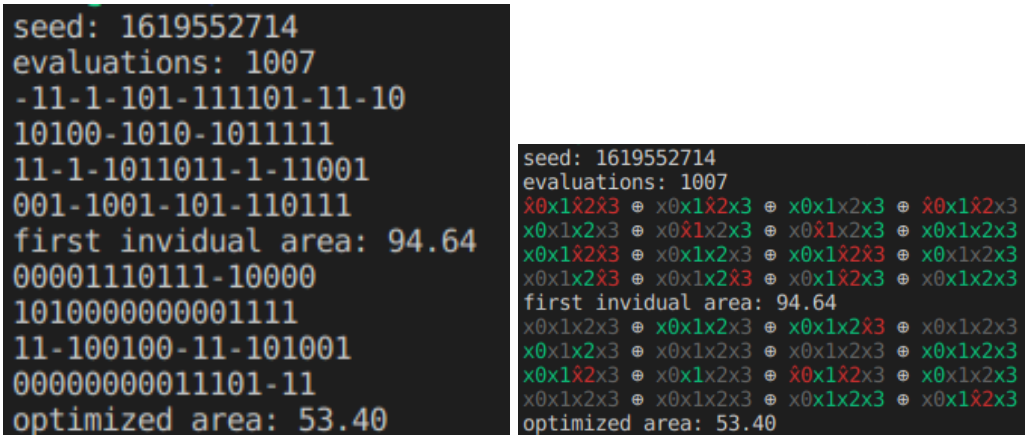


Obrázek 7.3: Diagram tříd s komponenty potřebnými pro obvod zakódovaný v ANF.

Třída Formula drží vektor Literálů, kde literál odpovídá vstupu X_i. Je potřeba rozlišit, jestli se podílí na výpočtu nebo nikoliv. Z tohoto důvodu literál může nabývat tří stavů:

- Not - odpovídá negovanému vstupu.
- IsNot - odpovídá neaktivnímu vstupu (nepodílí se).
- Is - odpovídá přítomnost vstupu.

Formule pak následně podporuje metody pro výpočet fitness (pouze daného řádku), vytisknutí formule, mutaci atd. Výstup programu je znázorněn na obr. 7.4. Barevné provedení (šedá - neaktivní prvek, červená - negace, zelená - vstup) je zejména vhodné při ručním ladění parametrů. Pokud by konzole nepodporovala barevné provedení, je zde možnost využít pouze výstup v podobě ascii znaků, který odpovídá přímo zakódování (viz sekce 6 popisující ANF chromozom).



(a) (b)

Obrázek 7.4: Obrázek odpovídá výstupu programu implementující ANF metodu s parametrem pro optimalizaci plochy (2x2 bit násobička). (a) Varianta zobrazuje výstup pouze v ascii znacích (b) Varianta zobrazuje barevný výstup.

7.8 Redukce plochy ANF

Jak bylo avizováno dříve, obvody reprezentované pomocí ANF je možné různorodě optimalizovat na velikost výsledné plochy. Já jsem se rozhodl pro relativně jednoduchou implementaci.

Negované literály jsou předsunuty mimo obvod, kdy v případě jednoho negovaného literálu vyskytujícího se napříč formulemi, tak se zajistí, aby bylo započtena pouze jedna negace vstupu.

V případě termů hledám stejný vzor zapojení:

$$\begin{aligned} y_0 &= x_1x_3 \oplus \bar{x_0}x_1x_3 \oplus x_0x_2 \oplus x_1x_2x_3 \\ y_1 &= x_1x_2 \oplus x_0x_2x_3 \oplus \bar{x_0}x_2 \oplus x_1x_2x_3 \end{aligned} \tag{7.1}$$

V tomto konkrétním případě 7.1, je negace stejného literálu $\bar{x_0}$ respektive vstup s indexem 0, ve dvou případech: v první formuli druhý term a druhá formule třetí term.

Pro termy je vytvořena společná množina, kde se ukládají indexy literálů. Začne se s prázdnou množinou, do které se vloží první sada čísel {1, 3} (formule y_0), poté následuje {0, 1, 3}, {0, 2}, {1, 2, 3}, takto se zpracuje první formule. Množina po prvním řádku vypadá následovně: {{1, 3}, {0, 1, 3}, {0, 2}, {1, 2, 3}}. Nyní se přidají indexy z druhého řádku (formule y_1), které jsou: {1, 2}, {0, 2, 3} a poté se narazí na {0, 2}, {1, 2, 3}, které již jsou v množině. Výsledná množina vypadá následovně: {{1, 3}, {0, 1, 3}, {0, 2}, {1, 2, 3}, {1, 2}, {0, 2, 3}}. Z takovéto množiny je vypočteno, že je použito 9 hradel AND (pro třívstupové hradlo je započtena dvakrát cena dvouvstupového hradla AND). Z množiny negovaných vstupů máme 1 hradlo NOT.

Kapitola 8

Experimenty

Bylo provedeno několik sad experimentů, které porovnávají běžně používanou variantu CGP s dvouvstupovými hradly, CGP s redukovanou množinou log. funkcí a ANF, u které bylo hlavním cílem zjistit, zda může přinést nějaké výhody oproti CGP.

Sady experimentů jsou rozděleny podle obvodů, které v nich byly navrhovány. Jejich výsledky jsou vzájemně porovnány a popsány včetně možného důsledku. Primárním ukazatelem je procentuální úspěšnost algoritmu v nalezení řešení ze všech 31 běhů, které vždy byly provedeny. Druhým ukazatelem účinnosti metod je počet evaluací fitness funkce nutných k nalezení plně funkčního řešení. Dalším ukazatelem je plocha, kterou navržené obvody potřebují k realizaci. V úvahu je brána plocha obvodu nalezení plně funkčního řešení, pak až plocha po optimalizaci.

U každého experimentu je krabicový graf (respektive grafy), který zobrazují minimum, 1. kvartil, medián, 3. kvartil a maximum ze zaznamenaných hodnot. V případech, kdy plně funkční obvod nebyl nalezen, nejsou tyto hodnoty v grafech již zobrazeny, aby byla nejdůležitější část výsledků čitelná. Na x-ové ose je uvedeno, o kterou metodu se jedná. První podgraf má vždy ještě u metody uvedenou procentuální úspěšnost.

Aby bylo možné lépe vybrat a porovnat chování jednotlivých parametrů, uskutečnil jsem značnou řadu různých nastavení, tak abych následně vybral nejlepší možné (viz přílohy). Pro výběr představených výsledků jsem se rozhodl na základě těchto kritérií:

1. Ideálně 100% úspěch nalezení plně funkčního řešení ve všech bězích.
2. Počet evaluací.
3. Plocha obvodu – první nález.
4. Plocha obvodu – optimalizovaná.

Protože byla vybrána široká škála obvodů, nebylo zcela možné sjednotit parametry pro všechny experimenty. Z tohoto důvodu je vždy pro každý konkrétní obvod seznam použitých parametrů.

V případě CGP (včetně redukované množiny) byl parametr L-back vždy nastaven na maximální hodnotu konektivity, protože toto nastavení usnadňuje celou evoluci. Parametr pro počet řádků je vždy pouze jeden řádek ($r = 1$ viz podsekce 5.1.1).

Jako ukončující podmínka evoluce sloužil počet generací. Velikost populace byla ve všech případech rovna $\lambda + 1$, kde $\lambda = 4$.

Níže jsou použity tři různé označení:

- ANF — obvody zakódované v ANF.
- CGP — obvody zakódované v CGP.
- CGP-R — obvody zakódované v CGP s redukovanou množinou funkcí.

Běžné CGP (v případě této BP) má množinu funkcí {IN, AND, OR, XOR, NOT, NAND, NOR, NOT XOR}, kdežto CGP-R má redukovanou množinu funkcí {IN, AND, XOR, NOT}, čímž byla snaha napodobit ANF. Hradlo *IN* pouze replikuje první vstup na výstup, tj. dá se chápat jako vodič.

8.1 Sudá parita 5b

Nejjednodušším z navrhovaných obvodů je sudá parita na 5ti bitech. Typicky tento obvod tvoří pouze funkce XOR, která se použije postupně na všechny bity. Na základě této znalosti není překvapením, že zakódování v ANF metodě měla 4krát rychlejší konvergenci (graf na obr. 8.1). To je značné urychlení oproti CGP metodě. Velikost plochy prvního plně funkčního obvodu si vedla ANF metodě zhruba dvakrát lépe, nicméně se projevuje fakt, že ANF je více restriktivní. Důsledkem je očividné uváznutí v lokálním extrému při optimalizaci plochy, kde CGP v každém běhu našlo minimální možnou plochu (5 hradel XOR). To bylo zapříčiněno i nastavením parametrů, kdy parametr *arity* = 1 způsobil rychlejší konvergenci, ale nedal žádný prostor pro stále funkční změnu.

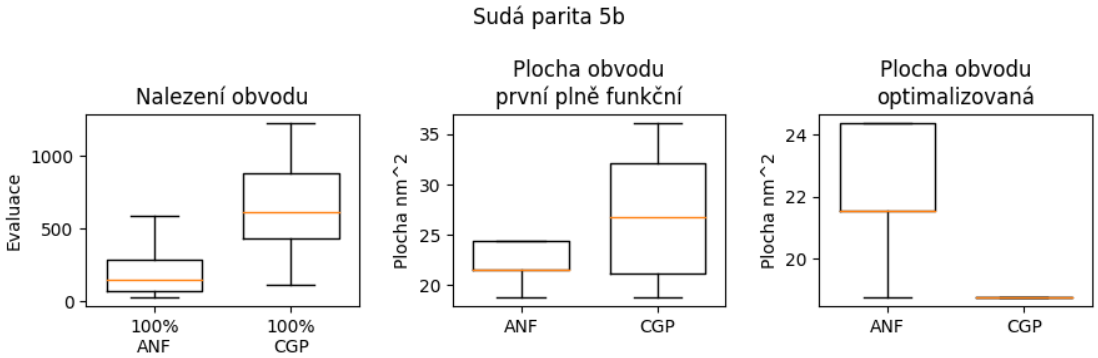
Ukázalo se, že ANF dokáže redukovat plochu obvodu obdobně jako CGP, pokud dostane dostatečný prostor (příloha A.1), byť za následek je vyšší počet evaluací pro nalezení plně funkčního řešení (prostor řešení je mnohonásobně větší).

Parametry ANF:

- Počet termů = 5
- Mutace = 2
- Arita = 1
- Počet generací = 5000000

Parametry CGP:

- Počet sloupců = 40
- Mutace = 8 %
- Počet generací = 1500000



Obrázek 8.1: Graf s výsledky pro sudou paritu 5b; 5 vstupů, 1 výstup.

8.2 Násobička 2b x 2b

Násobičky jsou jedním z nejkomplexnějších obvodů. Násobičky všeobecně jsou typicky tvořeny hradly AND, XOR a NOT. Proto se do jisté míry dá očekávat, že kandidátní řešení zakódované v ANF by mohlo najít plně funkční obvod rychleji. Tento předpoklad byl správný a oproti CGP byla konvergence zhruba 3krát rychlejší (medián).

Plocha na druhou stranu vlivem zakódování je asi dvojnásobná. Případ prvního nálezu funkčního nálezu (prostřední graf na obr. 8.2). Po následné redukci plochy, výsledky také nejsou zrovna nejlepší. Byť plocha se zmenšila asi o 50 %, tak obvody nalezené pomocí CGP byly 3 až 4krát menší, a to i v případě redukce plochy. To mohlo být zapříčiněno zvolenými parametry, protože na grafu v příloze A.2 je vidět, že téměř pro všechny nastavení se velmi povedlo zredukovat plochu.

Zajímavé jsou výsledky pro CGP-R, kde konvergence byla rychlejší jak v případě CGP (menší prostor kandidátních řešení). Plocha byla téměř identická s CGP.

Parametry ANF:

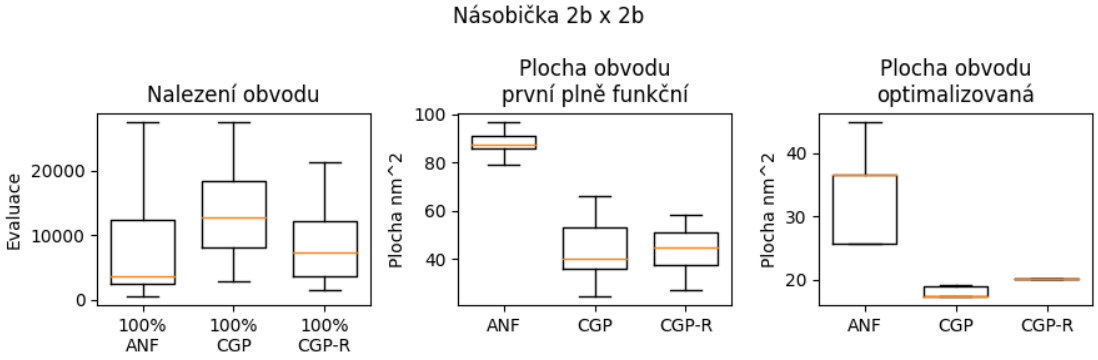
- Počet termů = 4
- Mutace = 4
- Arita = 4
- Počet generací = 1e+06

Parametry CGP:

- Počet sloupců = 50
- Mutace = 2 %
- Počet generací = 1e+06

Parametry CGP-R:

- Počet sloupců = 70
- Mutace = 2 %
- Počet generací = 1e+06



Obrázek 8.2: Grafy výsledků pro násobičku 2b x 2b; 4 vstupy, 4 výstupy.

8.3 Násobička 3b x 4b

Obě metody si v případě menších násobiček vedly dobře (viz 8.2). Každým přibývajícím vstupem se komplexnost zvyšuje. Což se projevilo i na výsledcích pro násobičku $3b \times 4b$, kde varianta ANF dopadla s 87% úspěšností, zatímco CGP pouze s 19%. CGP-R varianta dopadla ještě hůře viz obr. 8.3. Příčinou je nedostatečný počet generací. Z literatury [9] je známo, že pro složitější obvody je potřeba desítky milionů generací na nalezení plně funkčního obvodu ($3b \times 4b$ zhruba 25 milionů).

U varianta CGP-R se očekávalo, že dopadne obdobně jako v případě menší násobičky, což nenastalo. Příčinou může být, že se musí nalézt hradlo *XOR*. Tento fakt a nedostatečný počet generací poté vyústil ve špatný výsledek.

Plocha nalezených obvodů pro CGP (obdobně CGP-R) je asi 8krát menší. Ale výsledky nejsou moc relevantní vzhledem k malému počtu funkčních řešení. Podobná situace nastala i pro následnou optimalizaci plochy.

Vliv na úspěšnost a plochu (hlavně redukci) je možné vidět na grafu v příloze A.3. Dá se očekávat, že s větším počtem generací by redukce plochy mohla jít na hodnoty podobné CGP metodě.

Parametry ANF:

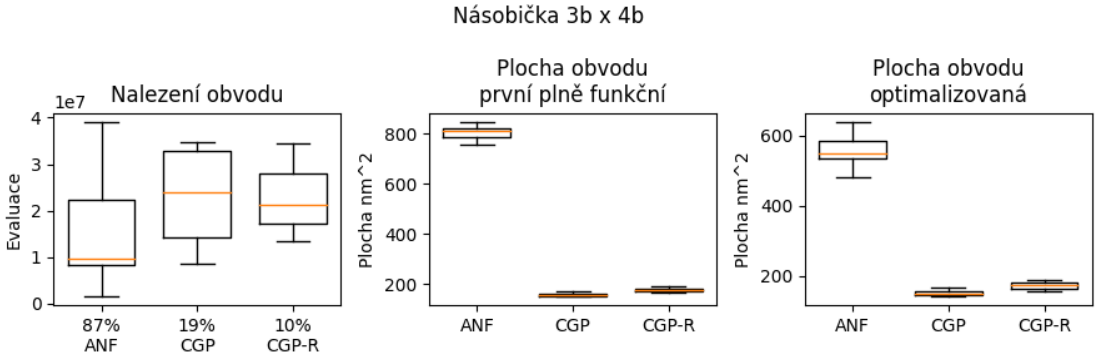
- Počet termů = 14
- Mutace = 2
- Arita = 7
- Počet generací = 1e+07

Parametry CGP:

- Počet sloupců = 60
- Mutace = 2 %
- Počet generací = 1e+07

Parametry CGP-R:

- Počet sloupců = 80
- Mutace = 2 %
- Počet generací = 1e+07



Obrázek 8.3: Grafy výsledků pro násobičku 3b x 4b; 7 vstupů, 7 výstupů.

8.4 Sčítačka 3b + 3b

U 3bitové sčítačky jak v CGP ani ANF nebyl žádný problém najít plně funkční obvod. Statistické vyhodnocení počtu generací potřebných k nalezení plně funkční-ho obvodu zobrazuje obr. 8.4. Nicméně z maximálního a průměrného počtu generací vyplývá, že CGP si vedlo lépe. Toto může být zapříčiněno tím, že u sčítačky je potřeba použít hradlo *OR*, které v ANF přímo není. CGP má přímo hradlo *OR* v množině funkcí.

Z výsledků, které jsou patrné z grafů znázorňující plochu obvodů (obr. 8.4) vyplývá, že kandidátní řešení nalezené zakódované metodu ANF jsou mnohonásobně větší, 7krát oproti CGP a CGP-R.

V příloze A.4 je možné pozorovat negativní dopad nevhodné volby hodnoty mutace.

Parametry ANF:

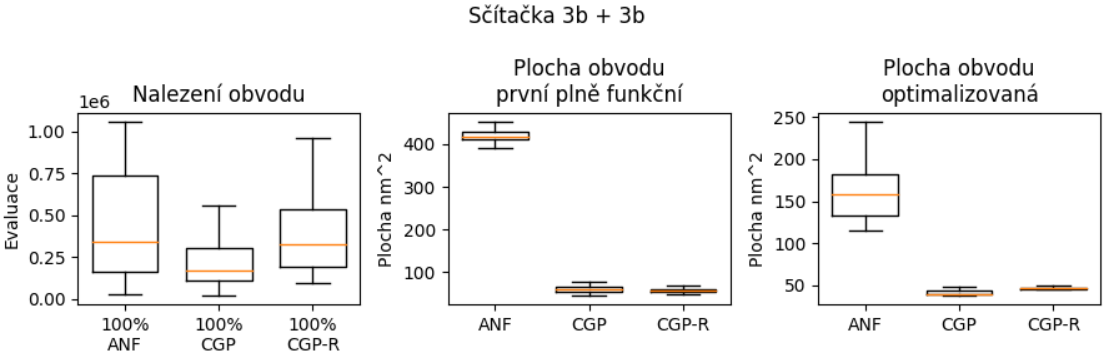
- Počet termů = 14
- Mutace = 4
- Arita = 6
- Počet generací = 4e+06

Parametry CGP:

- Počet sloupců = 50
- Mutace = 2 %
- Počet generací = 3e+06

Parametry CGP-R:

- Počet sloupců = 50
- Mutace = 3 %
- Počet generací = 3e+06



Obrázek 8.4: Grafy výsledků pro sčítačku 3b + 3b; 6 vstupů, 4 výstupy.

8.5 Řadicí síť 6b

Řadící síť je kombinační obvod, který se využívá pro řazení k-bitových vektorů do neklesající posloupnosti. Tento druh obvodu je snazší navrhnout pomocí CGP než násobičku se stejným počtem vstupů. Řadící síť je možné v evolučním návrhu nalézt pouze pomocí hradel AND a OR (AND hradlo se nahradí za funkci realizující minimum ze dvou hodnot, a OR realizuje maximum). Podle tzv. Zero-One teorém je zaručeno, že pokud síť dokáže seřadit všech 2^k vektorů, tak řadící síť se stejnou strukturou je schopna seřadit libovolnou posloupnost k-bitových vektorů [12].

Řešení zakódované pomocí ANF metody dosáhlo stejně jako CGP ve všech běžích plně funkčního obvodu. Plocha i po redukci zůstala víceméně stejná. Oproti CGP a CGP-R (s malou úspěšností nalezených řešení) byla plocha asi 4krát větší. Tím pádem řadící síť není zrovna nejvhodnější pro ANF kódování. Co do počtu evaluací, tak do velikosti plochy si vedlo CGP mnohem lépe.

V příloze A.6 je možné vidět, že s vyšším počtem termů klesá počet potřebných evaluací k nalezení funkčního řešení. Bylo by vhodné provést další řadu testů s vyšším počtem termů.

Parametry ANF:

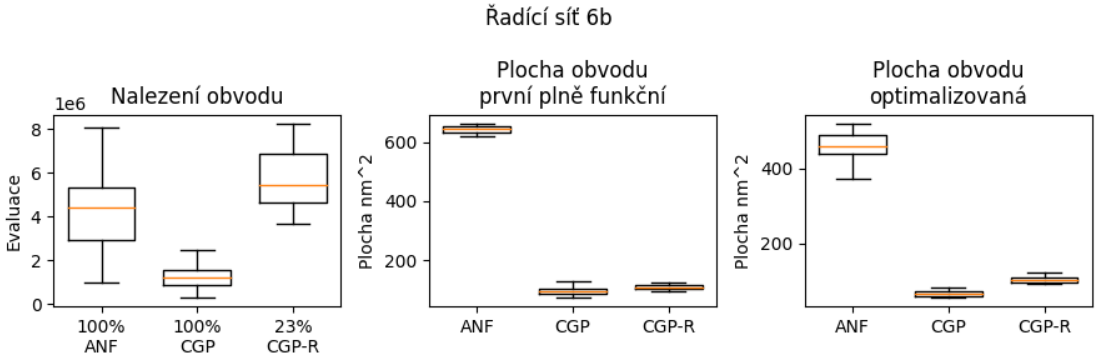
- Počet termů = 16
- Mutace = 2
- Arita = 6
- Počet generací = 4e+06

Parametry CGP:

- Počet sloupců = 80
- Mutace = 2 %
- Počet generací = 2e+06

Parametry CGP-R:

- Počet sloupců = 60
- Mutace = 2 %
- Počet generací = 2e+06



Obrázek 8.5: Grafy výsledků pro Řadící síť 6b; 6 vstupů, 6 výstupů.

8.6 Medián 5b

Seřadí-li se N čísel podle jejich hodnoty, tak prostřední číslo je mediánem vstupní posloupnosti. Platí, že polovina čísel posloupnosti má hodnotu menší nebo rovnu mediánu a polovina větší nebo rovnu. Pro sudý počet vstupů je potřebná dohoda, které číslo se bude považováno za medián.

Medián nachází uplatnění zejména při zpracování obrazu (filtrace šumu). Podobně jako u řadicí sítě i zde platí Zero-One teorém.

Ukazuje se, že medián zakódovaný pomocí ANF je z pohledu úspěšnosti evolučního návrhu nejvíce vhodný ze všech testovaných úloh. K nalezení řešení je třeba nejméně generací, a to téměř 15krát méně než v případě CGP a CGP-R. Bohužel na druhou stranu velikost plochy a následná redukce už není tak dominantní. Plocha je zhruba 3 až 4krát větší.

Zajímavostí může být, že CGP i CGP-R dopadly téměř s totožnými výsledky. Jak pro potřebný počet evaluací, tak i redukce plochy je totožná. Protože v množině funkcí není hradlo XOR bylo předpokládáno dosažení horších výsledků.

Parametry ANF:

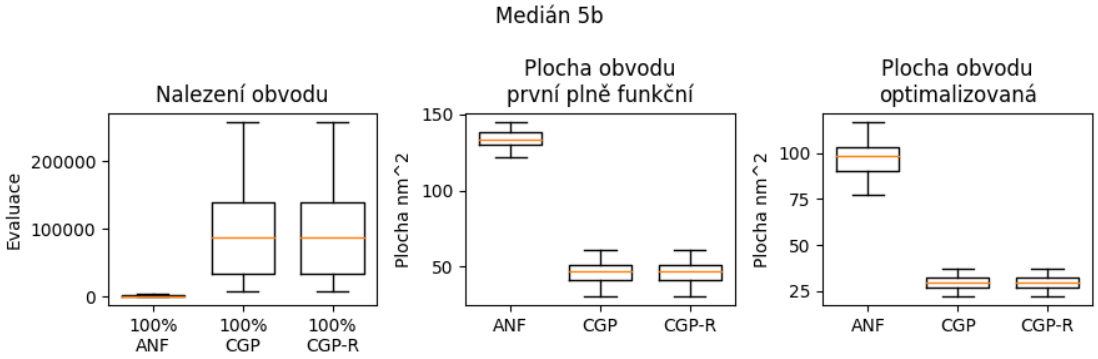
- Počet termů = 16
- Mutace = 2
- Arita = 5
- Počet generací = 1e+06

Parametry CGP:

- Počet sloupců = 50
- Mutace = 3 %
- Počet generací = 1e+06

Parametry CGP-R:

- Počet sloupců = 60
- Mutace = 2 %
- Počet generací = 1e+06



Obrázek 8.6: Grafy výsledků pro medián 5b; 5 vstupů, 1 výstup.

8.7 Komparátor 4b

Komparátor je kombinační logický obvod, který porovnává dva binární vektory a generuje výstup o 3 bitech – menší, rovnost, větší (rovnost nebo různost). Z hlediska použitých hradel v komparátorech je potřebná funkce nonekvivalence, též známé jako hradlo XOR. Společně s hradlem XOR je nutné mít ještě AND, NOT a OR hradla.

Obvody vyevolvované za pomocí kódování v ANF měly téměř 100% úspěšnost (kromě jednoho nepovedeného běhu). Nicméně z počtu evaluací (obr. 8.7) vyplývá, že mnohem vhodnější je použít CGP. Obdobné je to i u plochy, kde CGP mělo 8krát lepší plochu.

Úplně stejné výsledky platí pro CGP-R. Nutnost nalezení hradla XOR ztížilo prohledávání, a z tohoto důvodu dosáhlo CGP horších výsledků v počtu evaluací.

Parametry ANF:

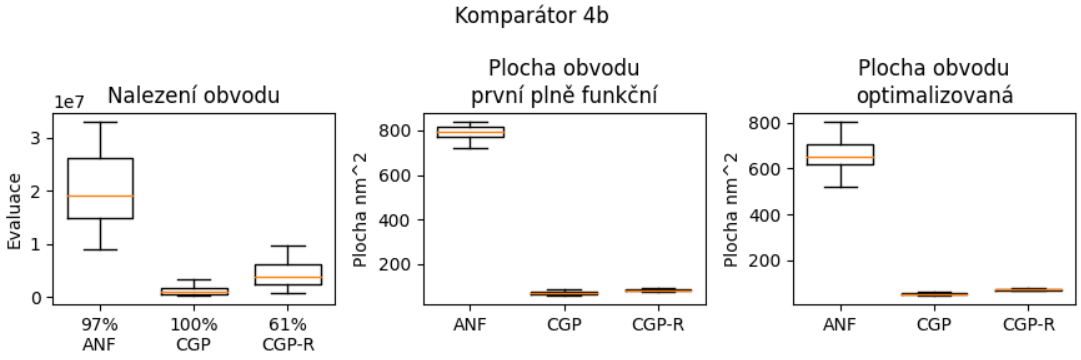
- Počet termů = 26
- Mutace = 4
- Arita = 8
- Počet generací = 1e+07

Parametry CGP:

- Počet sloupců = 50
- Mutace = 2 %
- Počet generací = 2e+06

Parametry CGP-R:

- Počet sloupců = 60
- Mutace = 3 %
- Počet generací = 2e+06



Obrázek 8.7: Grafy výsledků pro komparátor 4b; 8 vstupů, 3 výstupy.

8.8 Zhodnocení experimentů

Výsledky mluví jednoznačně ku prospěchu metody používající ANF, pokud hledáme obvody, u kterých je známý vysoký výskyt hradel XOR (případně AND a NOT). Počet evaluací potřebných k nalezení plně funkčních obvodů byl mnohonásobně nižší než v případě CGP. Ze sady testovaných obvodů se neukázala přímo nevhodnost ANF pro konkrétní případ.

Na druhou stranu, pokud chceme zredukovat nalezenou plochu obvodu, je mnohem vhodnější použít CGP, které dominovalo v každém z experimentů. Proto se naskýtá úvaha o propojení silných stránek obou metod. Jednou z možností je, že se prvně nalezne funkční obvod pomocí ANF, a poté CGP provede optimalizaci plochy. Tento způsob by se mohl výrazně projevit na počtu evaluací potřebných k nalezení složitějších obvodů, kterými jsou například $6b \times 6b$ násobičky, kde CGP naráží na své limity.

Kapitola 9

Závěr

V této práci byl představen základní princip fungování evolučních algoritmů, úvod do problematiky optimalizace kombinačních obvodů a následně princip metod zakódování CGP a CGP s redukovanou množinou. Dále byla vytvořena evoluční metoda využívající zakódování obvodu pomocí ANF.

Zejména byla popsána problematika vázaná s ANF (mutace, vyhodnocení fitness, zjištění plochy). Byly demonstrovány silné a slabé stránky obou metod, kde experimenty odhalily, že kódování metodou ANF může mít velký potenciál u násobičky a mediánu. Konkrétní vliv parametrů na prohledávání je možné nalézt v příloze. Experimenty zároveň odhalily, že reprezentace obvodu založená na ANF není vhodná pro optimalizaci plochy obvodu.

Další možnosti a vylepšení, které by bylo zajímavé prozkoumat, je urychlení návrhu použítím kombinací obou metod. Mohlo by být zajímavé porovnání s dalšími reprezentacemi obvodů (které jsou založeny na jiných formách), popř. rozšíření pro obvody s neurčitě zadanými vstupy.

Literatura

[1] *Karnaugh Maps, Truth Tables, and Boolean Expressions*. Dostupné z: https://instrumentationtools.com/topic/karnaugh-maps-truth-tables-and-boolean-expressions/.

[2] *Quine-McCluskey Tabular Method*. Dostupné z: https://www.tutorialspoint.com/digital_circuits/digital_circuits_quine_mccluskey_tabular_method.htm.

[3] Branke, J. *Evolutionary optimization in dynamic environments*. Vyd. 1. Boston: Kluwer Academic, 2002. ISBN 0-7923-7631-5.

[4] Chalupník, V. *Biologické algoritmy (3) - Evoluční algoritmy*. Internet Info, s.r.o., Duben 2012. Dostupné z: https://www.root.cz/clanky/biologicke-algoritmy-3-evolucni-algoritmy/.

[5] Eiben, A. E. a Smith, J. E. *Introduction to Evolutionary Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015. Natural Computing Series,. ISBN 9783642072857.

[6] Jasper. *Logické obvody 3 - Normální formy a Karnaughovy mapy*. Dostupné z: https://www.itnetwork.cz/hardware-pc/hardware/logicke-obvody-normalni-formy-a-karnaughovy-mapy.

[7] Lameres, B. J. *Introduction to Logic Circuits & Logic Design with VHDL*. Cham: Springer International Publishing, 2017. ISBN 9783319341941.

[8] M., W. *Multimodal landscape*. Jul 2015. Dostupné z: http://www.wilsonmongwe.co.za/understanding-the-em-algorithm/.

[9] Miller, J. Cartesian genetic programming: its status and future. *Genetic Programming and Evolvable Machines*. Springer New York LLC. 2019, sv. 21, 1-2. ISSN 13892576.

[10] Mrazek, V., Sys, M., Vasicek, Z., Sekanina, L. a Matyas, V. Evolving Boolean Functions for Fast and Efficient Randomness Testing. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. New York, NY, USA: Association for Computing Machinery, 2018, s. 1302–1309. GECCO '18. DOI: 10.1145/3205455.3205518. ISBN 9781450356183. Dostupné z: https://doi.org/10.1145/3205455.3205518.

[11] Pták, O. *Evoluční resyntéza kombinačních obvodů*. Vysoké učení technické v Brně. Fakulta informačních technologií, 2013.

[12] Sait, S. M., Abd El Barr, M., Al Saiari, U. a Sarif, B. A. B. *Fuzzified Simulated Evolution Algorithm For Combinational Digital Logic Design Targeting Multi-objective Optimization*. Dostupné z: https://bit.ly/3bswT4b.

[13] Sekanina, L., ed. *Evoluční hardware : od automatického generování patentovatelných invencí k sebemodifikujícím se strojům*. Vyd. 1. Praha: Academia, 2009. Gerstner. ISBN 9788020017291.

[14] Vašíček, Z. *Biologií inspirované počítače - kartézské genetické programování*. 2021. [Online] Dostupné z http://www.fit.vutbr.cz/~vasicek/courses/bin_lab1/.cs. Dostupné z: http://www.fit.vutbr.cz/~vasicek/courses/bin_lab1/.cs.

[15] Češka, M., Matyáš, J., Mrázek, V., Sekanina, L., Vašíček, Z. et al. Adaptive verifiability-driven strategy for evolutionary approximation of arithmetic circuits. *Applied soft computing*. Elsevier B.V. 2020, sv. 95. ISSN 1568-4946.

Seznam příloh

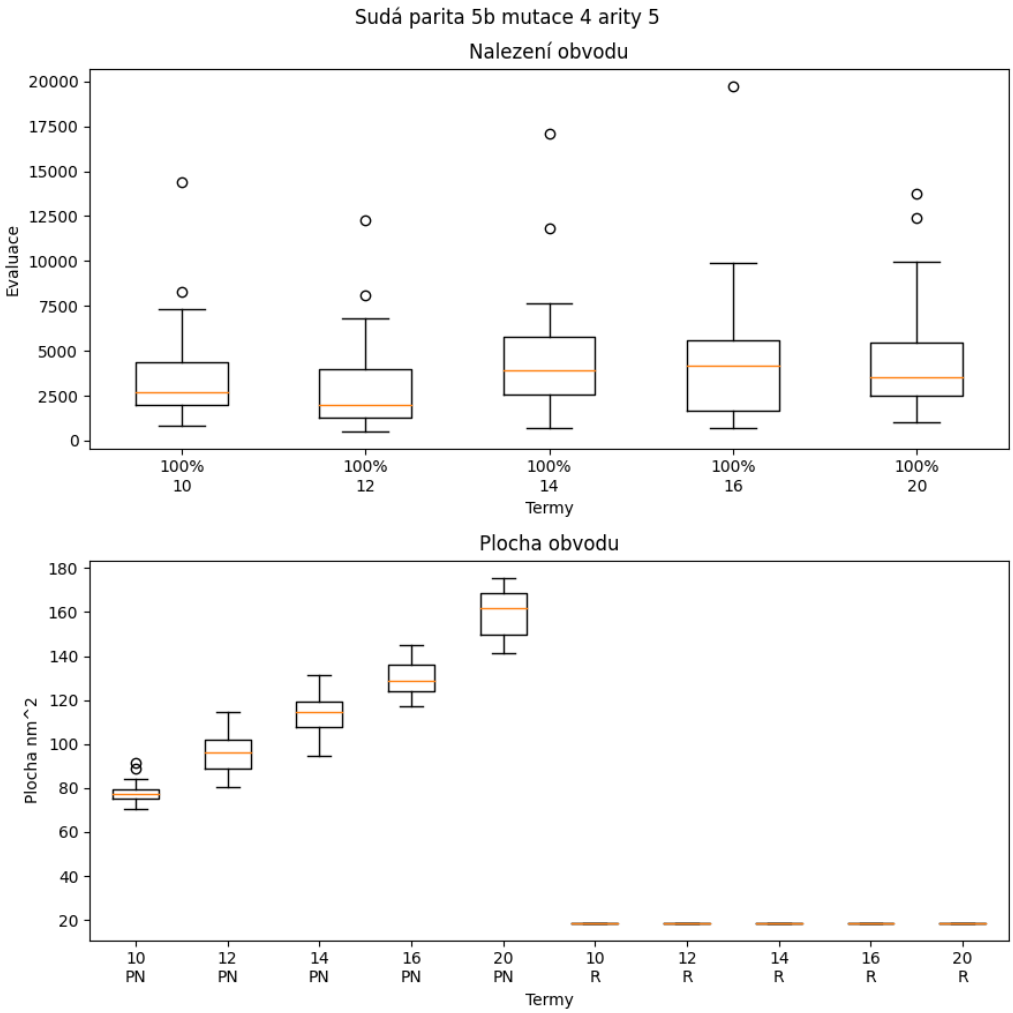
Přílohy / Appendices

Příloha A

Grafy naměřených hodnot

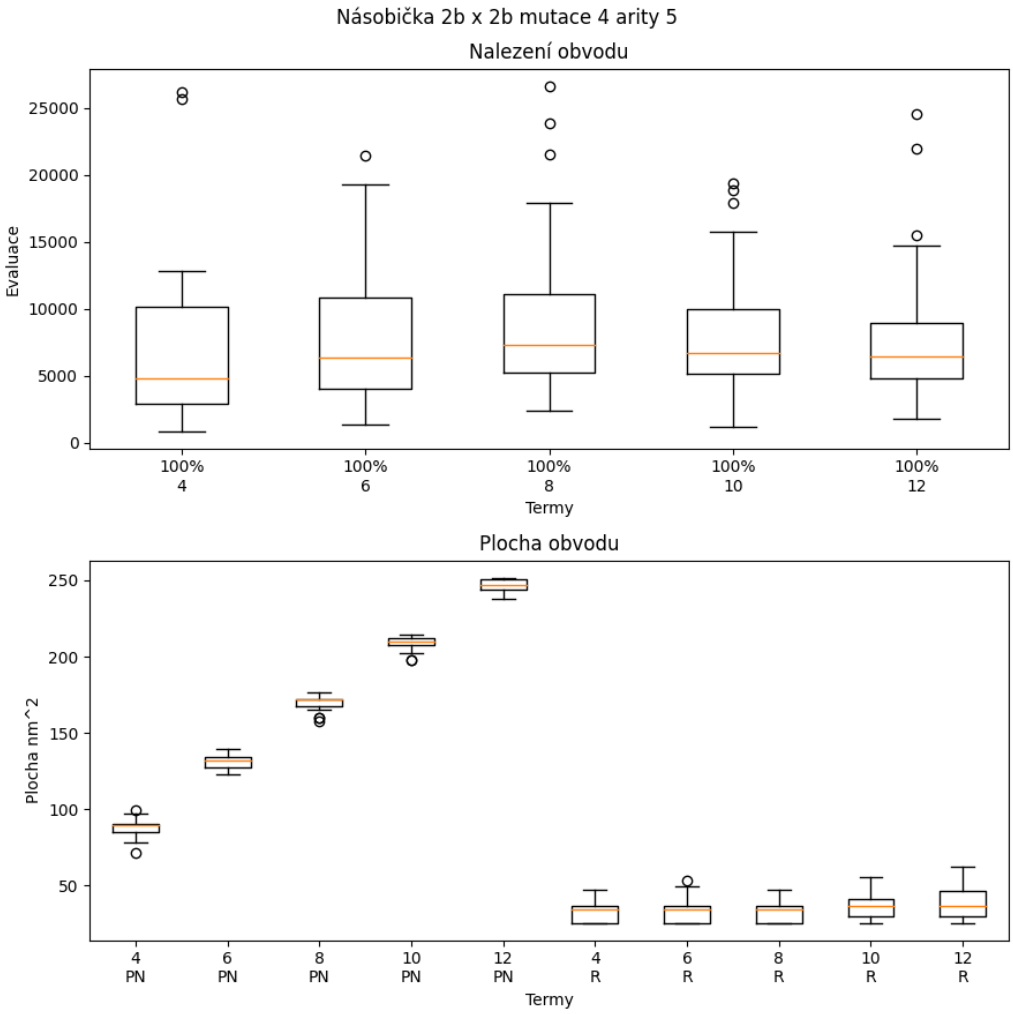
Parametry jsou vždy uvedeny v hlavičce grafu. PN – První Nalezený plně funkční obvod. R – Redukovaná plocha (po optimalizaci).

A.1 Sudá parita 5b



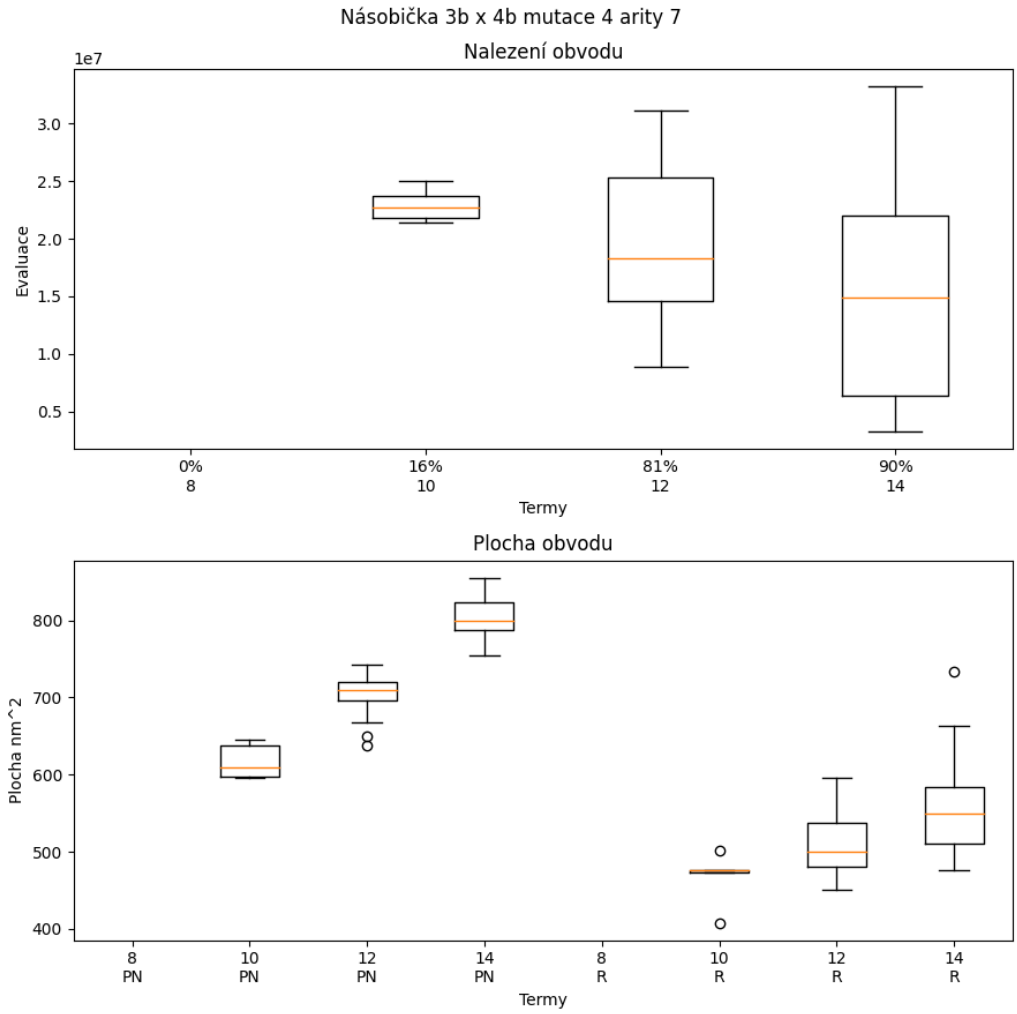
Obrázek A.1: Vliv parametrů pro metodu ANF.

A.2 Násobička 2b x 2b



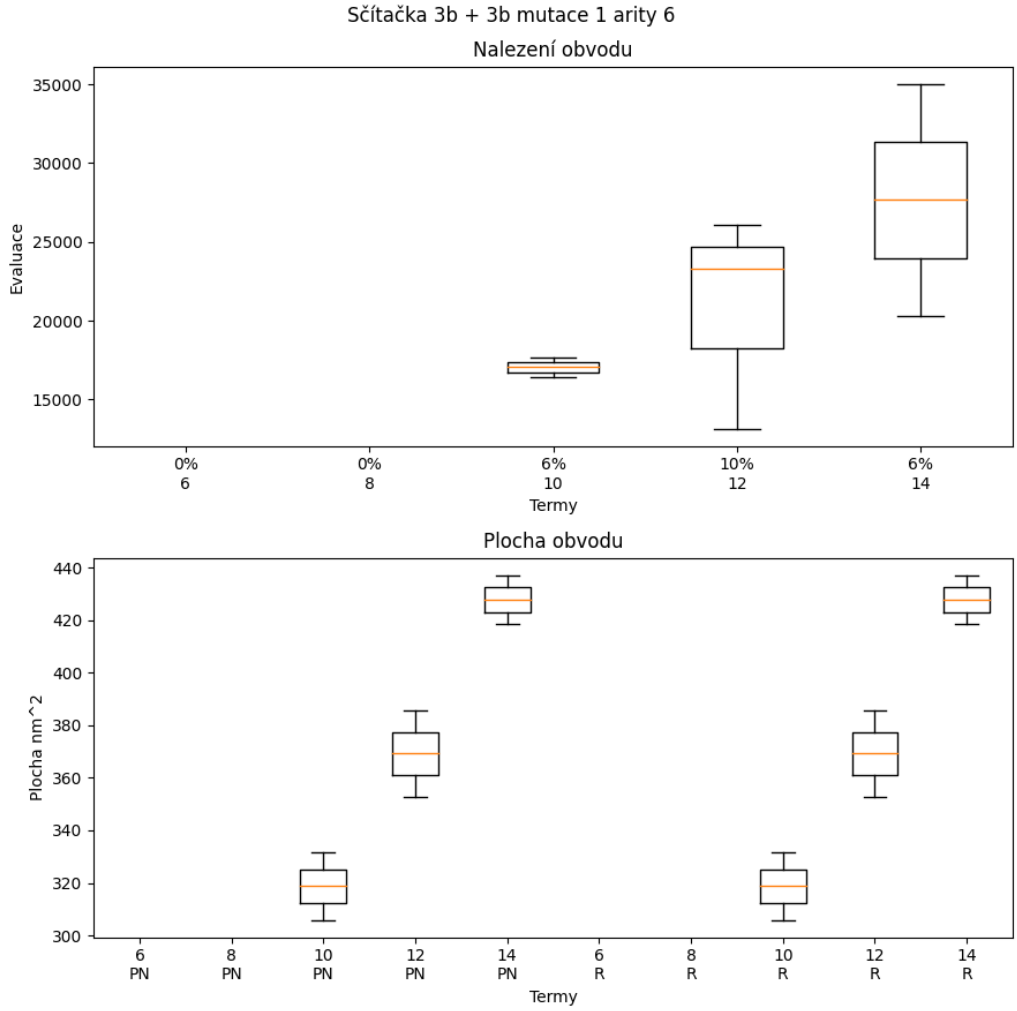
Obrázek A.2: Vliv parametrů pro metodu ANF.

A.3 Násobička 3b x 4b

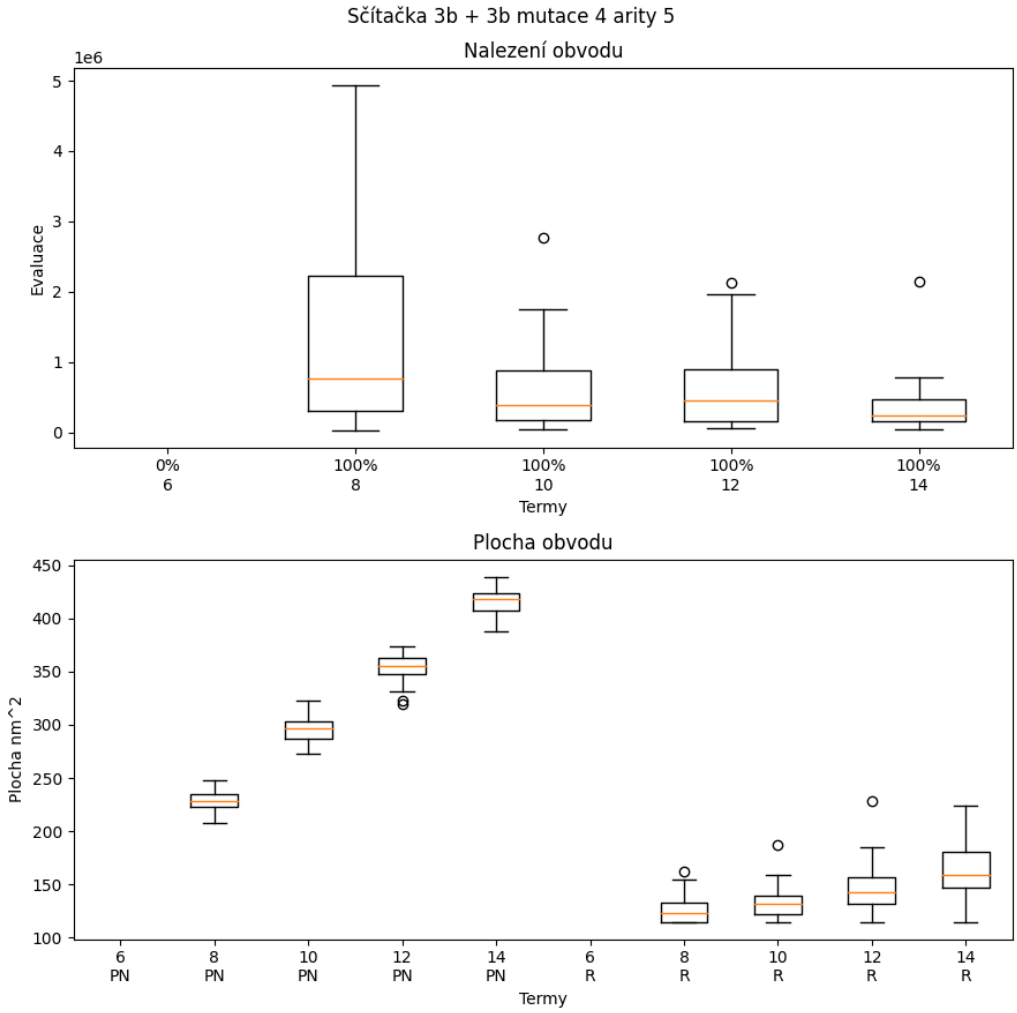


Obrázek A.3: Vliv parametrů pro metodu ANF.

A.4 Sčítačka 3b + 3b

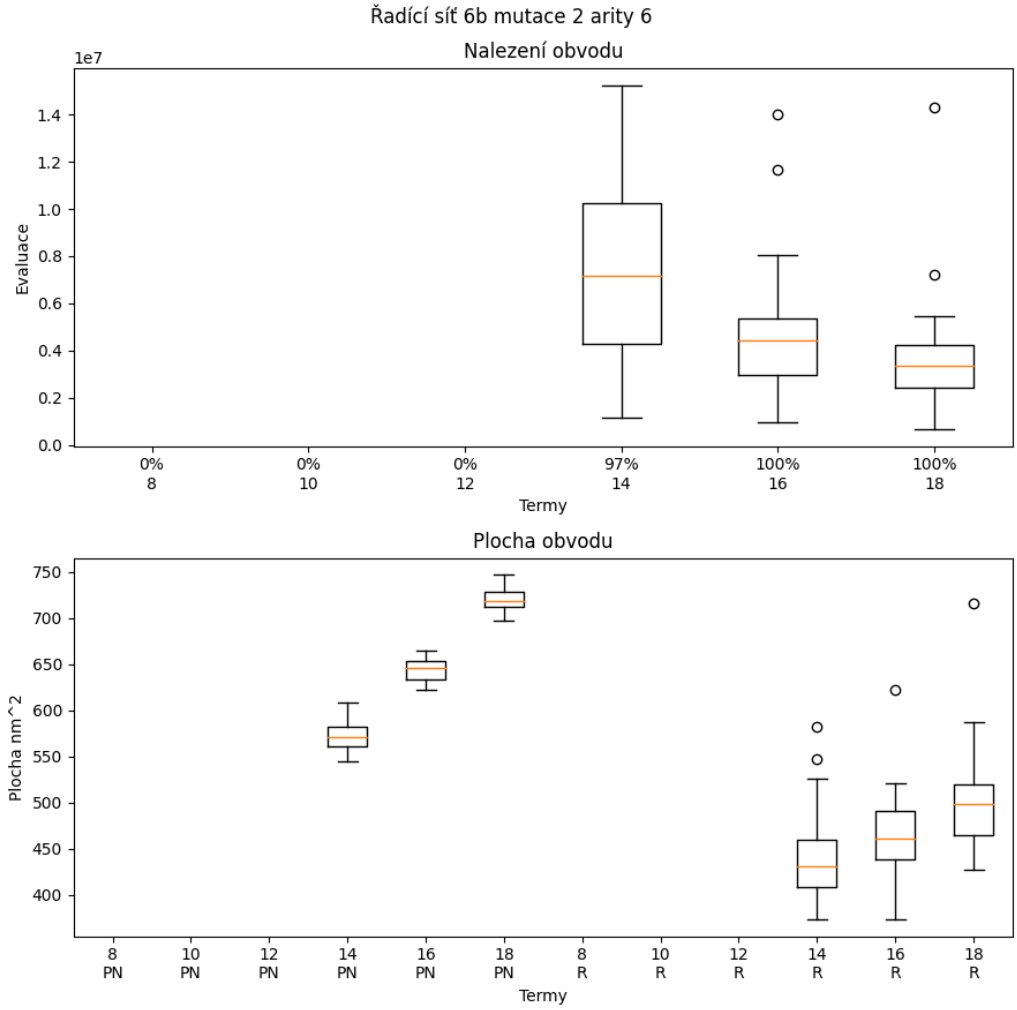


Obrázek A.4: Vliv parametrů pro metodu ANF s nevhodnou volbou mutace.



Obrázek A.5: Vliv parametrů pro metodu ANF s vhodně zvolenou mutací (vliv na plochu).

A.5 Řadící síť 6b



Obrázek A.6: Vliv počtu termů v metodě ANF.